FT-100

PROGRAMMABLE FUNCTIONAL TESTER

OPERATOR'S MANUAL

Y-tek, Inc.

Every effort has been made to ensure that this manual is error free. If you should find any errors, we would appreciate it if you would notify Y-tek, Inc. We would also appreciate any input or suggestions for improvement. Although every effort has been made to eliminate errors, Y-tek cannot assume responsibility for any errors in this manual or their consequences.

FT-100a Operator's Manual, First Edition 1998
Part Number: 983-0100-01

Y-tek, Inc.

2634 Pleasant Union Church Road
Raleigh, North Carolina 27614
Telephone (919) 676-4741
www.Y-tek.com

Table of Contents

1.0 Introduction

1.1 Product Description

The FT-100a is a programmable functional tester designed to operate independently of any other computers or controllers and be affordable to even the smallest manufacturing, design, or depo repair company. It contains many features found in much larger and more expensive systems. Most other automatic test equipment requires additional input/output "units" or "boards" to build a functional test system capable of handling even relatively simple testing. These combined modules must then be configured to operate with each other to perform the required duties. While the FT-100a has the capability of expanding its input/output capabilities through external I/O modules, the base unit provides enough I/O to handle most of a small manufacturer's testing needs.

Anyone unfamiliar with the FT-100a should thoroughly read this entire manual. After achieving an understanding of the overall operation, the manual will be used mainly as a reference tool. The section on Software Commands will be particularly useful during test program development.

1.2 Features

Hardware Interfacing

The base unit of the FT-100a is equipped with the following interfaces:

- +5 volts @ 1 amp Power Source (overcurrent protected)
- +12 volts @ 1 amp Power Source (overcurrent protected)
- -12 volts @ 250 ma. Power Source (overcurrent protected)
- Programmable power source (0-10 volts) @ 1 amp (overcurrent protected)
- 40 Digital Input lines (Five 8-bit Ports; one port with external trigger capabilities)
- 40 Digital Output lines (Five 8-bit Ports)
- 8 Analog Input lines (0-10 volts or +/-5 volts)
- 7 Analog Output lines (0-10 volts or +/-5 volts)

The digital input lines are HTCMOS type with 47k ohm pull-up resistors and switch at standard TTL input levels. The digital output lines are high current CMOS type and provide a rail-to-rail output (0-5 volts). Both the digital input and output lines are directly accessible through dedicated software commands, either as individual lines or as 8-bit ports. Analog I/O, along with the programmable power source, is also directly accessible through dedicated software commands. A thorough description of the input/output capabilities and requirements may be found in the Hardware Interface section.

Memory and Data Storage

All internal RAM memory is battery-backed. This eliminates the daily requirement of reloading a program that is used on a continuing basis. While data may be collected in internal RAM, it may be more permanently stored on a floppy disk. This also permits much larger amounts of data to be collected. Collected data may also be printed on an RS-232 serial printer or sent to another computer.

Program Loading and Storage

Programs may be loaded from a floppy disk or through the serial I/O port. Since the disk drive is IBM PC compatible (3.5 inch, 1.44M), programs may be developed on an IBM compatible computer (using any word processor program or text editor) and loaded into the FT-100a through the disk drive.

Direct Control

In order to permit real-time operation and control of the FT-100a, the user may take complete control through the RS-232 port. Most program commands as well as some special "Direct" commands are available for real-time operation. This can be very beneficial during troubleshooting defective units, as well as during program development. In some instances, the operator may execute "direct" commands while a program is running.

Optional Keyboard

The operator may use the optional FT-100a keyboard to interface with the test program for data entry or to answer direct questions. Direct commands may also be entered using the keyboard. The keyboard provides an excellent method of entering such data as serial numbers, production run numbers, etc.

Expansion Capabilities

Although the FT-100a main unit contains significant Input/Output capabilities, whenever the need arises for even more or different I/O, optional Expansion Modules may be added at any time, with no modifications required. Expansion Modules are separately housed, and connect to the FT-100a through a dedicated expansion connector on the rear of the tester. Expansion Modules are available for almost any conceivable I/O requirement. Contact Y-tek for the latest information on available Expansion Modules.

1.3 How It Works

For the FT-100a to perform its test and/or control functions, it should be connected to the unit to be tested (or controlled) through the user interface connectors on the front panel. There must also be a program installed in the FT-100a to tell it what you want it to do and how you want it to do it. The Hardware Interfacing section thoroughly explains the interface conditions and limitations and the Programming section will guide you through the design of test programs.

Once the unit is ready to begin operation, simply press START/STOP on the front panel to start the program (this task may also be accomplished by typing "START" or "RUN" on the keyboard or a terminal or computer connected to the serial interface port. This is described in detail in the Direct Command Mode section). The program may also be started by activating the START/STOP line in the I/O interface (on the front panel). When the program is started, the FT-100a will perform the duties determined by the installed program. Upon completion of the program, the display will indicate that the program has finished.

2.0 Operation

2.1 Initial Check-out

After unpacking the FT-100a, check to see that there is no shipping damage and that all accessories are included. If there is obvious shipping damage, contact the shipper immediately. If any accessories are missing, please contact Y-tek Customer Service immediately.

At this time connect the power cord to the back of the unit and plug into a power source of 100-240v. AC, 50/60 Hz. Do not connect any other cables to the FT-100a (front or back). Turn on power by switching the rocker switch on the front panel. The front panel display should light and the FT-100a will go through a brief self-test. If the FT-100a finds any internal problems during the self-test, it will so indicate (See appendix for error messages and suggestions).

After a successful self-test, the display should indicate:

```
Y-tek       FT-100a
Prog: {prog name}
```

(The last program loaded into the FT-100a will be indicated in place of {prog name}.)

Now that you are assured the FT-100a is functioning, it's time to become familiar with its operation.

2.2 Front Panel Controls

Following is a brief description of each of the FT-100a front panel controls:

POWER - Turns all power on and off. The green LED inside the POWER switch will indicate when power is on.

START/STOP pushbutton - When pressed, the loaded test program will immediately begin execution. If pressed during program execution, the program will immediately stop. This control is also used for some of the menu selections. (See START and STOP in Reference Section for more details)

WAIT/CONT pushbutton - If a program is running, pressing this pushbutton will cause the program to "pause" after the execution of the current command. When pressed again, the program will continue where it left off. This control is used for some menu selections as well. (See WAIT and CONT in Reference Section for more details)

COMMAND pushbutton - There are several operating commands accessible after pressing this pushbutton. Most of these commands allow control of the disk drive. When pressed, the command menu will be indicated on the front panel display.

ESC (escape) pushbutton - Some of the internal modes and routines may be aborted by pressing this pushbutton. Almost all disk drive functions use this as an abort mechanism.

2.3 Power-on Setup

There are some default parameters within the FT-100a that may be set through an initial setup routine. While most of these parameters may be set through program commands, sometimes it is necessary to initialize to a known state. Several parameters influence the serial communications protocol and must be set before serial communications can be established.

There are two ways to get into the Power-on setup routine: (1) Turn on power while holding the START/STOP pushbutton, or (2) Press the RESET pushbutton on the back panel while holding the START/STOP pushbutton. The FT-100a will sound a "beep" and the display will show:

```
SET-UP See Manual
  > BAUD: 4800 <
START: to accept
CONT: to change
```

Pressing the START/STOP pushbutton will accept the indicated selection. Pressing the WAIT/CONT pushbutton will scroll to the next selection. Use care in making the selections. If a wrong selection is made, simply turn power off, or press RESET, and go back into the beginning of the setup routine again. To get out of the setup routine without going through all setup selections, press the ESC pushbutton.

Follow the instructions as you go through the setup selections. Following is an explanation of each of the setup parameters:

- Baud Rate - The serial data communications rate. Choose the baud rate that your communications device can support. It is normally better to choose the highest baud rate that both communicating devices can comfortably handle.

- Echo (on/off) - When Echo is <u>on</u>, each character sent to the FT-100a through the serial communications port is "reflected" and sent back out the serial port. (The only exception to this is during a program LOAD. Characters are never "echoed" during a program LOAD).

- Auto Linefeed (on/off) - When enabled, Auto LF will automatically place a linefeed character after every carriage return. This applies to all data sent out through the serial communications port.

- Printer Width - The FT-100a uses the serial port for printer output. Setting the "Printer Width" determines the number of characters that will be sent before a carriage return (and possibly linefeed) will be sent. If a serial printer is not to be used, select N/U (not used).

- System Initialization - A full system initialization will go through the FT-100a internal operating system and initialize all internal parameters to a know state. This is sometimes necessary when service to the FT-100a has been done or the user has any doubt about the integrity of the FT-100a setup. On any unit that has been shipped, it is always a good idea to go through a full system initialization. Sometimes, due to abnormal program halts or power interruptions, the internal pointers can be left in undefined states. The only sure way to reset these internal pointers is through the System Initialization.

CAUTION: SYSTEM INITIALIZATION WILL ERASE EVERYTHING IN MEMORY, INCLUDING THE PROGRAM!

2.4 YES/NO Pod

To allow direct operator interface with a running program, the YES/NO pod may be used to answer simple yes/no questions. The words, "YES" and "NO" are keywords and may be read in the program like variables (they cannot be set, however). When either YES or NO is read, the value will be one (1) if the corresponding pushbutton is pressed "at that instant"; otherwise, the value will be zero (Ø). YES and NO can be used in conditional statements or even in expressions, just like any other variable.

It is very important to remember that there is no "latching" of the YES and NO pushbuttons. In other words, to read the pushbutton correctly, it must be pressed at precisely the instant that it is read. Normally, a program loop is used when reading the YES or NO pushbuttons.

2.5 Hardware Interface

In order for the FT-100a to perform any testing or control functions on another unit, the hardware interface between the two must be established. This is done simply by choosing the FT-100a I/O interface lines needed and connecting from the I/O interface to the unit to be tested and/or controlled. All hardware interface (input and output) is done through the three connectors across the front of the FT-100a.

Following are the input, output, control and power supply lines available to the user through the I/O connectors, along with a description of each:

I/O Interface Lines

These are the lines used to drive the unit being tested/controlled or to "read" output information.

- Digital Input lines - HTCMOS inputs (TTL input levels) with internal 47k ohm pullup resistors. Use these lines to monitor any 5 volt logic levels. These lines are configured as 8-bit digital input ports. Each 8-bit port has its own unique software reference allowing 8 digital lines to be treated as one entity. Individual lines may be accessed through unique software references as well.

- Digital Output lines - CMOS outputs (Ø-5 volts) with 220 ohm current-limiting resistor in each line. Use these lines to drive any CMOS compatible input lines (see the Hardware Interface reference for drive capabilities). These output lines are also configured as 8-bit digital output ports, and may be referenced in the same way as the input lines.

- External Trigger line - HTCMOS input (TTL input compatible) with 10K ohm pullup resistor. The FT-100a main unit has one 8-bit digital input port (address PA10) that has a programmable trigger. Through a software command, the polarity of the trigger activation may be set (low-to-high or high-to-low). Once the polarity is established and the gated port is reset, the proper transition on the external trigger line will cause the digital input port to latch at its current state. The port can then be read through software. The input port will remain latched until reset by the program command.

- Analog Input lines - These lines are used to monitor and read any DC voltage within the selected input range. Through a program command, the voltage range may be either 0-10 volts or +/-5 volts. All analog input lines have 12 bits of resolution. Each analog input line has its own unique program reference. (Expansion Modules have different specifications)

- Analog Output lines - (Ø-10 volts). The analog output lines may be set, through program commands, to any voltage within the selected analog range. Use these lines any time a known, variable, or setable DC voltage is needed. (Expansion Modules have different specifications)

Supply Lines

The following are Power Supply lines available to be used to power the unit being tested/controlled. Each line has a thermal, self-resetting current limiter. Neither temporary nor permanent overloads should harm the FT-100a. When overloaded, the voltage will drop to a very low level, limiting the output current to the maximum specified level. After removal of the load, the power supply will reset itself. However, due to the thermal nature of the overload protection, it may take a few seconds for the supply line to reset.

Available Supply Lines at I/O Interface:

- +5 volts @1 amp - Use to power any 5 volt logic or linear circuits. This is the same supply voltage used for the FT-100a logic. Although not always necessary, circuits run from this supply will be voltage compatible with the FT-100a logic I/O.

- +12 volts @1 amp

- -12 volts @250 ma.

- Variable Voltage Supply (Ø-10 volts) @1 amp - This supply line's voltage is set by its own software command (SUPPLY). Even though this is a power supply line, it may be used like any other analog output line.

- Digital Ground - Although not normally considered a supply line, the ground lines carry the same current load as the supply lines and act much like other supply lines. All the FT-100a input/output logic is referenced to this ground, and the unit being tested/controlled should as well. All power and logic should use these lines as the ground return. There are several ground lines within the I/O interface connectors, and all within any used connector should be tied to the unit under test. *CAUTION - Internally these lines tie to the metal cabinet and earth ground through the AC power cord.*

- Analog Ground - These lines connect to Digital Ground internal to the FT-100a, but because of the sensitivity of analog signals to digital noise, there is a separate ground return for analog signals (input and output). Many times it is impossible to use separate grounds for digital and analog signals, but these lines are available and should be used whenever possible for all analog ground returns. If the unit being tested/controlled has no analog input or output needs, the analog ground lines may be tied to the digital ground lines.

Control

These control lines allow access and control of the FT-100a basic operations through the I/O connector.

- START/STOP - Active low, CMOS input with 10K ohm pullup resistor. This line duplicates the function of the front panel START/STOP pushbutton. Whenever the START/STOP line is "pulled low", the FT-100a will react as if the START/STOP pushbutton was pressed.

- WAIT/CONT - Active low, CMOS input with 10K ohm pullup resistor. This line duplicates the function of the front panel WAIT/CONT pushbutton. Whenever the WAIT/CONT line is "pulled low", the FT-100a will react as if the WAIT/CONT pushbutton was pressed.

- RESET - Active low, CMOS input with 10K ohm pullup resistor. When the RESET line is pulled low, the FT-100a is forced into a "hard" reset. This places the FT-100a in the same condition as after power-up or pressing the rear panel RESET pushbutton. After a catastrophic problem, the FT-100a may need to be reset.

Precautions and Suggestions

- Try to use all supply and ground lines available to minimize ground loop "noise" problems and decrease any voltage drop through the interface. If no analog lines are being used, tie the analog ground lines to digital ground.

- All I/O ground lines ultimately tie to earth ground through the FT-100a. There are no floating ground lines.

- Be aware that there is normally digital "noise" throughout most digital circuits. If the circuit being tested has a low tolerance for noise on the interface cables, steps must be taken to either lower the noise or isolate the circuitry. Bypass capacitors on the interface can usually help this problem, but may slow down the reaction time. If noise is observed on any of the interface lines (at the unit being tested), place a 0.01 µf capacitor on the offending lines near the sensitive circuitry. For more serious noise problems, larger capacitors may be place across the I/O lines, but be sure to account for any delays that might result.

- Analog lines are particularly susceptible to noise. Always use good grounding techniques and isolate the lines as much as possible. Try to keep digital and analog circuitry separate, if possible.

- If the unit being tested does not have good power supply filtering, depending on the circuits involved, extra bypass capacitors might need to be added to the power supply lines.

Note: See <u>Hardware Interfacing</u> in the <u>Appendix</u> for more information about hardware interfacing requirements, precautions, mechanical configuration and pin-out of the I/O connectors, etc..

2.6 Keyboard (optional)

The FT-100a has an interface connection on the rear panel of the unit for a standard AT-type keyboard. With a keyboard connected, the operator may issue Direct Commands or input data directly into a test program (See "Direct Mode Operation" for detailed information about Direct Commands). All commands and modes of input that are available through the serial communications port are also available via the keyboard input (See <u>Serial Communications</u> for more information).

The INKEY and GETKEY commands are used by the test program to input data from the keyboard. A full explanation on the use of these commands can be found in the <u>Reference Section</u>.

2.7 Serial Communications

All serial communications with the FT-100a is done through the serial communications port located on the rear panel of the unit. Standard RS-232 protocol is used, as well as a standard DB25 interface connector. Serial communications is always half-duplex with RTS/CTS handshaking used on outgoing data. No handshaking is used on incoming data.

The FT-100a serial communications port has multiple uses. A serial printer may be attached for hard-copy printout, or a data terminal (or computer) may be connected for real-time communications or program/data transfer. By connecting a data terminal or computer to the serial communications port, the user may remotely control the FT-100a, load and list programs, execute program commands in real-time, and transfer data (both directions). Although not required for most operations, the FT-100a can often be more efficient when real-time communications is available. Nearly all aspects of control and programming are available through the serial communications port. Up to four (4) separate serial ports are available by connecting an optional 1041 serial port expander (available from Y-tek).

Communications Protocol - The following protocol is used in The FT-100a serial communications:

- FT-100a configured as DTE (Data Terminal Equipment)
- RS-232 format
- 8 Data Bits
- 1 Start Bit

- 2 Stop Bits
- No Parity
- RTS/CTS Handshaking (outgoing only)

For more detailed information on serial port interfacing, see Serial Communications in the Appendix.

Printer Output

The FT-100a uses the serial communications port for its printer output. If a hard-copy printer is to be used, connect an RS-232 serial printer to the serial port on the back panel. Please note, while a printer is connected to the FT-100a, there can be no real-time control or communications through the serial port unless an optional serial port expander is used.

Once the printer is connected, go through the power-on setup routine (see Power-on Setup) to set the baud rate to that of the printer. This may also be done by using the COM command (see Program Commands in the Reference Section). Auto Linefeed and Printer Width may need to be set as well (consult the printer manual for requirements). Now, all data sent out the serial port will be sent to the printer.

Terminal Communications

> For the sake of clarity, it will be assumed that all serial communications are done with a computer running a terminal emulation program. Exceptions will be noted.

Almost any data terminal or computer running a terminal emulation program can communicate with the FT-100a. The communications protocol is standard ASCII, and any program that can emulate an ASCII or TTY terminal should work fine (VT-52 or VT-100). Consult the manual for setup information.

Normally it is best to choose the highest baud rate that both devices can support. Because the FT-100a does not acknowledge XON/XOFF, a slow device that cannot support RTS/CTS handshaking may not "catch" everything at very high baud rates. If it looks like some data is missing, try increasing the character delay (using the COM command), or use a slower baud rate. The FT-100a should have no trouble receiving all supported baud rates.

Making the Connection

Step 1 - Connect RS-232 cable between the serial port on the back of the FT-100a and the serial port of the computer (or terminal). See Serial Communications in the Appendix for cable specifications.

Step 2 - Set up the computer (or terminal) communications protocol. Consult the appropriate manual to set these parameters. If using a computer, be sure it is emulating an ASCII terminal, and go into Local mode. Set the communications parameters to the following:

Baud Rate: 19.2k or 9600 (Others may be used, but it will be slower).

1 start bit, 8 data bits, 2 stop bits

No Parity

No Automatic Linefeed after carriage return.

Step 3 - Set up the FT-100a communications protocol. (Through the Power-on setup routine.)
Set the FT-100a protocol to the following:
Baud Rate: Same as Step 2.
Echo: ON
Auto LF: ON
Printer Width: N/U (set only if automatic wraparound is not supported)

Step 4 - Be sure the computer/terminal power is on. Press RETURN several times on the computer. If communications is established, the terminal screen should show the character ">". If communications is not established, go through the procedure again. If you still have trouble, check that the cable is connecting the proper pins at each end, and the RTS and CTS lines are connected together at the FT-100a end (pins 4 & 5).

Step 5 - Communications is now established and the FT-100a may be issued commands through the serial line. Nearly all program commands, as well as Direct commands, are available through the serial communications line. (Try some simple commands, such as CLS or BEEP)

For more information on interfacing to the serial port, see Serial Communications in the Appendix.

Loading Programs via the Serial Port

The FT-100a can load in, and print out, programs using two different formats: ASCII text, and Compacted. The format for each program file self-identifies, so the program loading procedure is the same for both. An understanding of each type is beneficial, and a brief description follows:

ASCII program file - Simply a text file that contains the program (with no extra control characters). The format of the text is covered in detail in the Programming Section and the Appendix. An ASCII program file is created by a word processor or text editor program and is normally used when loading programs for the first time. When the FT-100a loads in an ASCII program, it will immediately process it into the more efficient "Compacted" format.

Compacted programs - The ASCII program is "processed" by the FT-100a and reduced to a form that provides the fastest execution and uses the least amount of memory for program storage. This form is the most efficient to load, and alleviates the need for the FT-100a to process the incoming program.

To load a program through the serial port, first be sure communications is established between the computer/terminal and the FT-100a. Then type "LOAD" followed by a carriage return (press ENTER on the computer). The FT-100a will "beep", and the display will indicate, "Loading ...". The FT-100a will now wait indefinantly for the program file to be sent through the serial communications line.

Each terminal and computer program has its own procedure for sending an ASCII file. Consult your manual for the procedure to follow for your system. Be sure to select "ASCII text" (or equivalent) as the data format. Begin sending at any time.

When the program file has been successfully transferred, the FT-100a will "beep" and the display will indicate "** READY **" and show the program name. The program is now loaded and is ready to be run.

2.8 Loading Programs from Floppy Disk

> Saving program files on floppy disks is covered in the DIsk Operation Section.
> For now, it will be assumed that the program is already on the floppy disk.

To load in a program from a floppy disk, first insert the floppy disk with the desired program into the disk drive until it latches. The disk LOAD command may be accessed either by a direct command through the keyboard or serial port, or through the front panel pushbuttons. Direct command loading will be covered in the Disk Drive Operation section, so for now we will concentrate on the front panel controls.

To access the disk commands from the front panel, press the COMMAND pushbutton. The display will indicate:

```
Command:   LOAD
START-  to select
 CONT-  for other
        commands
```

Each time the CONT pushbutton is pressed, the display will scroll through the available commands. If you wish to access the indicated command, press START; otherwise, press CONT to go to the next command. Pressing the ESC pushbutton will abort out of any disk command. To load in a program, select the LOAD command by pressing START at the above screen. The display will show:

```
  {program  name}
START-  Load Prog

 CONT-  Next Prog
```

The first program on the floppy disk will be indicated. Pressing CONT will scroll through all the program files. When the desired program is shown, press START to load the program. The program will be loaded automatically. If the program is in the ASCII text format, it will be processed and compacted as it is being loaded.

After the program has been loaded, the FT-100a will "beep" and indicate that it is "READY". The new program is now ready to be run.

2.9 Program Execution

There are several commands that affect program execution. Each of these commands may be initiated either from a front panel pushbutton, a control line in the I/O Interface, or a Direct Command through the keyboard or serial port. More information about Direct Commands may be found in Direct Mode Operation or the Reference Section. The Program Execution commands are:

START - Immediately starts the loaded program.

Direct Command Syntax: "START" or "RUN"

When a program is started, all digital outputs are set low (Øv.), analog output range is set to 0-10 volts, and analog outputs set to 0 volts. Any "open" disk files will be "closed", and all variables eliminated.

(If issuing commands through the keyboard or serial port, the commands, "START" and "RUN" are equivalent and may be used interchangeably)

STOP - Immediately stops the program.

Direct Command Syntax: "STOP"

Note: Whenever the STOP command is issued, the FT-100a will immediately stop all program execution, even in the middle of a command operation. The STOP command should be judiciously used since internal data and interface outputs may be in unknown states. If a disk file has been "opened", it will be "closed" when a STOP command is issued.

When a program ends, or a STOP command is issued, all communications parameters are reset to their values prior to the start of the program and all I/O outputs remain at their last states! This includes the variable voltage supply. Once the program has been halted using the STOP command, there is no way of restarting the program "where it left off".

WAIT - Causes a running program to pause after the current command or statement and sound a "beep". If a program is not running, WAIT has no effect.

Direct Command Syntax: "WAIT"

(WAIT is also a program command)

Note: Since the WAIT command does not interrupt a command in progress, data integrity is not compromised. When the program is paused with the WAIT command, all variables and I/O conditions are temporarily suspended and maintained. Using direct mode operation (through the keyboard or serial port), variables, I/O conditions, etc. may be observed and even altered while in a "wait" condition.

When a program is in a "wait" state, all communications parameters are temporarily set to their values prior to the start of the program. If a serial port expander (1041) is attached, communications port #1 is automatically selected. When the program is restarted by a CONT command, the communication parameters are set back to their values just prior to the WAIT command.

Note: Placing a WAIT command within your program is one way to indefinitely pause the program while the operator performs other duties (adjustments, measurements, etc.).

CONT - Causes the program to continue if it is currently in a "wait" state by a previously issued WAIT command. The program will continue with the next command/statement after the "wait" command was issued. If the program is not in a "wait" state, CONT has no effect.

Direct Command Syntax: "CONT"

Note: If any communication parameters were changed while the program was in the "wait" state, they will be reinstated to their previous values.

3.0 Programming (How to)

The FT-100a has its own very powerful programming language. Programming the FT-100a is very similar to programming in the BASIC computer language. Many of the commands and statements are the same. While experience in programming is always a plus, even a beginner, with little or not programming experience, should be able to construct FT-100a programs in short order.

One of the FT-100a's greatest assets is its unique language that permits direct addressing of input and output lines. As will be seen, I/O lines and ports are normally treated like variables. The simplicity will become obvious.

It is obviously beyond the scope of this manual to thoroughly teach all aspects of programming. But, with the information given, and some practice, most people should be able to handle most programming tasks. For more thorough instruction on programming, there are many good books available for the BASIC programming language. The FT-100a language is so similar that a fair knowledge of BASIC, in addition to this manual, will be all that's needed.

Before starting to program the FT-100a, it is recommended that each programmer, regardless of experience level, read and understand all the programming commands as well as the "direct commands". Most are similar to BASIC, but there are some differences and many improvements.

3.1 Program Structure

Programs are written, and executed, one statement at a time. Line numbers are not required and are ignored when the ASCII program is loaded in. Each line must be followed by a carriage return (this is usually the ENTER key on the terminal or computer). When the program is started, it will begin executing the first statement. Upon completion of the first statement, it will immediately begin the next, and so on until the end of the program or an END statement is reached. More than one statement may be placed on a line, separated by a colon (:). If there is more than one statement on a line, the first statement is executed, then the next, and so on. The exception to this is the IF-THEN statement (see the <u>Reference Section</u>).

The FT-100a is a very powerful computer, but it will do *only* what you tell it to do; no more, and no less. Each statement will be executed exactly as it is written. Syntax (the grammatical structure of each statement) is <u>very</u> important and cannot normally be changed. Be sure to use the <u>Reference Section</u> for information on each command and statement.

ASCII Program Structure

All programs must be loaded into the FT-100a the *first* time as an ASCII program. This is a text file that has been generated using a word processor or text editor program. The ASCII program file must be pure ASCII character text with no control characters. The ASCII program structure is as follows:

Line #1 - {Program Name} This is the name of the program and must be 8 characters or less. The program name must be on a line by itself (with a carriage return). A comment may be on the Program Name line if desired.

Line #2 - (program lines) Body of the program

.

.

Line #n-1

Line #n - "/" The <u>last line</u> of the ASCII program <u>must</u> consist of the slash character, "/" followed by a carriage return. No other characters are allowed on the same line. This is an end-of-file marker.

Example:

```
DEMO1                   'Program name
'
        X=0                                 'Beginning of body of program
BEG:    CLS: DISP "Value of X = ";X
        DISP "Press CONT": WAIT
        INC X: GOTO BEG
        END                                 'End of body of program
' Program body followed by "/"
/
```

Once the ASCII program has been successfully loaded into the FT-100a, it will be processed and compacted, and then may be SAVED as a "compacted" program. The compacted format is smaller, more efficient, and loads faster.

If a program is to be edited, even after it has been "compacted", it may be LISTED as an ASCII program. This will regenerate an ASCII file from the compacted format and send it out the serial port. Line numbers will be placed on each program line, but the FT-100a does not require line numbers when loading in an ASCII program.

3.2 Syntax Rules

Program Name - In the ASCII form of the program, the first line <u>must</u> be the program name, with no extension. The program name must start with an alphabetic character and be no more than 8 characters.

End-of-program marker - The slash ("/") character is used to mark the end of the program in the ASCII form. The slash <u>must</u> be on a line by itself, followed by a carriage return.

Comments - All text following an apostrophe (') will be ignored. A Comment may be on a line by itself, or it may be on the same line, but follow a command or statement. Note: After program compaction, all comments are eliminated.

Example:

```
'This is a comment line by itself
VAR1 = 3.4    'Comment on line with statement
```

Labels - Used to mark a given program line. Labels must be the first element in a program line and must be followed by a colon (:). Labels must begin with an alphabetic character A - Z and may be up to six (6) characters long. Numbers and the underline character "_" may be used within a label. If there is more than one statement on a line, only the first statement can have a label. Labels are used to indicate a location that may be referenced by another statement.

Length of Program Line - 128 characters maximum. This is true, even after the program has been compacted. Normally, the line length should be no problem, however if "BUFFER OVERFLOW" errors are encountered, try breaking the long line into two shorter lines.

Keywords - Words the FT-100a recognizes as having special significance. The programmer is not allowed to use keywords for any other purposes. All program command and statement words are keywords. See the <u>Appendix</u> for a list of keywords.

Spaces - Used to separate keywords, numbers, variable names, etc. In most cases, delimiters or TAB characters may serve as separators as well. Extra spaces are ignored, unless they are within quotes.

Example:

```
DISP VAR1                'Space separating a keyword
DISP 3.4+VAR1,VAR2       'Delimiters serve as separators
```

Tabs - Tab characters are treated much the same as "spaces". Tabs may be used to separate keywords or indent program lines. They may also be used to space out a comment that is on a program line.

Variables - Word symbol used to reference a changeable value. Variables names must begin with an alphabetic character A - Z and be no more than eight (8) characters in length. Variable names may contain numbers.
Variables are assigned a value in an equate statement (see Equate below).

Numerical variables may represent numbers in any format (Integer, floating point, hexadecimal, binary, exponential, etc.).

String variables are used to represent strings of ASCII characters and may be up to 128 characters in length. String variable names must end with the dollar sign, "$".

```
Example:  VAR1, VAR2, ABC, X        (Numerical variables)
          ABC$, NAME$, A$           (String variables)
```

Array Variables - Numerical variables that allow several variables to be accessed and processed together as one entity. Arrays can be multi-dimensional. The total number of elements in an array is the mathematical product of the maximum number of elements in each dimension. Array syntax consists of the array name followed by the dimensions in parenthesis, with no space between. Array variables, like other variables, are assigned a value in an "equate" statement (see Equates).
Array elements may be numerical variables or expressions.
Arrays must have memory set aside prior to their use. See DIM in Reference Section.

```
Example:  A(2)         '1 dimensional array
XYZ(2,8,4)             '3 dimensional array
ARRAY(J,K)             'Array with variables as dimensions
```

Strings - Series of ASCII characters. All strings must be enclosed with quotes. A string may contain any combination of characters, each represented by a number 1 - 255. This includes all the ASCII characters as well as control and special characters. The exception to this is the "double quote" character ("). Since this character is used to indicate the end of strings, the FT-100a will interpret it as the string end.

Equate - Statement that assigns a specific value to an entity (variable, output port or line, etc.).

```
Example:  VAR1 = 3.4 + (VAR2/8)     'Setting a variable
          LA34 = 1                  'Setting a digital output line
```

I/O References - There are two types of I/O references: Direct and Indirect.
The Direct reference refers to a specific I/O line or port by a numerical number, whereas the Indirect reference uses a parameter as a reference (much like an array).
Outputs are set like variables, with an "equate" statement.
(For more information, see Addressing I/O or the Reference section).

```
Example:  PA6 = 5           'Direct reference to port# 6
          DISP PA(VAR1)     'Indirect reference to port# VAR1
```

Parenthesis - Used to determine the order of evaluation of an expression. Any number or level or parenthesis is permitted. There must be an equal number of beginning and ending parenthesis.

```
Example:  VAR1 = ((VAR2+3.4)/(VAR3-(4.5+VAR4))-12.345)/VAR5
```

Functions - Command statement that operates on its parameters. All parameters must be enclosed by parenthesis. Exception: If there is only one parameter following the function keyword, the parenthesis may be omitted. Syntax details for each function may be found in the Reference Section.

Example: VAR1 = INT (3.4/VAR2)
VAR1 = INT 7.8 'Only one parameter

Number Formats - Numbers may be input or output in any of several forms:

Integer - A whole number, positive or negative, between the range of -32768 and 32767.
(Note: In the FT-100a, all integers are 16 bits long, including the sign bit.)
Example: 45, 19876, 0, -73

Floating Point - Any number, positive or negative, that has a fractional amount or is outside the range of -32768 to 32767.
(Note: In the FT-100a, floating point numbers have 23 significant bits with a 7-bit binary exponent)
Example: 3.14159, 6.0, -2.41

Exponential (Scientific Notation) - Mantissa, followed by "E", then the Exponent (power of 10).
Examples: 4.5E-3 'Same as 0.0045 or 4.5×10^{-3}
-1.469E2 'Same as -146.9 or -1.4×10^{2}

Binary - Base 2 representation of number, followed by "B".
Examples: 1101B 'Same as 13_{10}
101100010B 'Same as 354_{10}

Hexadecimal - Base 16 representation of number, followed by "H". Since hexadecimal format has 16 digits, the letters A - F are used to represent digits 10 - 15. Hexadecimal numbers beginning with a letter (A-F) must be preceded by a zero.
Examples: 3AH 'Same as 58_{10}
0FF0H 'Same as 4080_{10}

NOTE: The next formats are Output formats only, and are set by either the COM statement, or within a PRINT or DISP command.

Binary-8 (Also know as B8) - Least significant eight binary bits. All other bits are truncated (dropped).
Example: If VAR1 equals 763_{10} (1011111011B) then when printed or displayed using B8 format, result is 251_{10} (11111011B).

Hexadecimal-8 (Also known as H8) - Least significant two hexadecimal digits. All other digits are dropped.
Example: If VAR1 equals 763H, then when printed using H8 format, result is 63H.

Decimal-8 (Also known as D8) - Decimal representation of the 8 least significant digits of a number.
Example: Printing 1708 in D8 format yields 172. (1708 = 6ACH. Using the 8 least significant binary bits yields 0ACH, which is 172.)

Exponential-8 (Also known as E8) - Exponential representation of the 8 least significant bits of a number.

Note: When a number is to be displayed or printed, but for any reason, cannot "fit" into the designated display format, the number will be followed by an exclamation mark (!).

Specific syntax rules regarding program commands and statements may be found in the Reference Section.

3.3 Operator Types

Arithmetic Operators - Mathematical functions: Add, Subtract, Multiply, Divide.
The result of an arithmetic operation is always a floating point number.
Example: VAR1 = 3.4 + VAR2 / 9.6

Logical Operators - Boolean operators: AND, OR, NOT, NAND, NOR, XOR.
Performs the referenced Boolean function on the two elements and returns the logical result.
Note: All Logical operators have equal weight in the hierarchy of operators. (See Operator Hierarchy)

Logical Operators:

AND	OR	XOR
0 AND 0 = 0	0 OR 0 = 0	0 XOR 0 = 0
0 AND 1 = 0	0 OR 1 = 1	0 XOR 1 = 1
1 AND 0 = 0	1 OR 0 = 1	1 XOR 0 = 1
1 AND 1 = 1	1 OR 1 = 1	1 XOR 1 = 0

NAND	NOR	
0 AND 0 = 1	0 OR 0 = 1	NOT 0 = 1
0 AND 1 = 1	0 OR 1 = 0	NOT 1 = 0
1 AND 0 = 1	1 OR 0 = 0	
1 AND 1 = 0	1 OR 1 = 0	

Note: Logical operators may also be used to combine expressions and comparison statements. Examples: IF A=B AND C>D THEN GOTO L1
IF (A=5 OR B=6) AND C=7 THEN X=1

Comparison Operators - Compares the numerical values of two operandi.

Equal =	Not Equal <>
Greater than >	Greater than or equal >=
Less than <	Less than or equal <=

Note: Equal (=), in addition to being a comparison operator, is also used to set entities.

Examples: VAR1 = VAR2+5.6
IF VAR1 > 3.4 THEN GOTO LABEL1

Operator Hierarchy - Order in which operators within an expression are evaluated. In the absence of parenthesis, all operators have a hierarchial "weight", causing operations to be performed in a given order. The hierarchy, in descending order is:

Multiply, Divide (highest priority)
Add, Subtract
Comparison Operators
Logical Operators

Operators with equal hierarchial weight are evaluated on a left-to-right basis.

String Operators - Strings may be joined, end to end, by using the plus (+) operator.
Example: A$= "ABC" + "DEF" Result is: A$= "ABCDEF"

Strings may be compared using the Equal operator. When compared, the strings must be *exactly* alike in order to be equal. This includes the length and inclusion of spaces.

```
Examples:   IF "ABC" = "ABC" THEN GOTO LABEL1      'Strings equal
            IF "ABC" = "ABC " THEN GOTO LABEL2     'Not equal
```

3.4 Addressing I/O

One of the most powerful features of the FT-100a programming language is the ability to address and process input/output information just like a variable. This makes programming extremely simple and versatile. Syntax information on I/O addressing can be found under Program Commands in the Reference Section. Information on available I/O and physical connections may be found in the Hardware Interfacing section in the Appendix.

CAUTION: When a program ends, or a STOP command is issued, all outputs will remain at their last states (including the variable voltage supply).

I/O Address designators

I/O lines are addressed through a "self-writing" designator. Any exception to this will be covered under the individual I/O type descriptions.

Structure of I/O address: {I/O type}{module}{reference}

{I/O type} - Following are accepted I/O types:		
	P - Digital Port	TP - Trigger Polarity
	L - Digital Line	LR - Latch Reset
	A - Analog line	LS - Latch Status

{module} - Module designator. The main unit is always module "A". The external modules are designated as B - H. The module designator is individually set on each external module through dip switches or jumpers.
Under no condition can any two modules share the same module designator.

{reference} - The reference refers to the individual I/O line within the module and may be a number, a variable, or a numeric expression. If the reference is a number, there should be no space between the module designator and the number reference. If the reference designator is a variable or expression, it must be enclosed within parenthesis (just like arrays).

Following are examples of I/O references:

PA4	- Digital Port, Module "A", Port# 4
AC5	- Analog line, Module "C", line# 5
LRE(VAR1)	- Latch Reset, Module "E", reset# designated by variable VAR1

Digital Ports

All digital ports consist of eight lines which may be addressed individually (see Digital Lines) or as an 8-bit port. The ability to input, output, and process port information as one entity can be very convenient. Additionally, the FT-100a has several commands that allow bit manipulation.

All digital port references begin with the letter "P", followed by the module designator (A-H), then the port# reference (either a number or expression).

Available Digital Ports in Main Unit:

PA1, PA2, PA3, PA4, PA5	Output Ports
PA6, PA7, PA8, PA9, PA10	Input Ports (Note: PA10 is an externally triggered port as well)

Reading and Setting Digital I/O Ports - Digital I/O ports are handled just like variables. (Attempts to set an input port will result in an ERROR, however no physical damage will occur.)

Examples:	PA1 = 3AH	'Set output port #1 in main module to 3AH
	PC4 = VAR1	'Set output port PC4 to value in variable VAR1
	PF2 = VAR2 / (PA3 + 3.4)	'Treat digital ports just like variables
	PA(VAR1) = PA6 + PD4	'Referencing a port# by a variable
	IF PB3 > 35H THEN BEEP	'Port data tested like a variable

Digital Lines

Each digital port is made up of eight individual lines. Each line may be addressed separately, allowing individual control of any single digital line.

All digital line references begin with the letter "L", followed by the module designator (A-H), then the line reference. Digital lines may be referenced in one of two ways: (1) line number, or (2) port number and line number (1-8) within the port, separated by either a comma or colon. See Program Commands in Reference Section for further information.

Available Digital Lines in Main Unit:

	LA1, LA2, ·····, LA40	
or:	LA1,1, LA1,2,·····, LA1,8, LA2,1, ·····, LA5,8	Output Lines
	LA41, LA42, ·····, LA80	
or:	LA6,1, LA6,2, ·····, LA6,8, LA7,1, ·····, LA10,8	Input Lines

Reading and Setting Digital I/O Lines - Digital I/O lines are also handled just like variables. Reading a digital line will yield a value of either "1" or "Ø". If a digital line is set to any value other than zero (Ø), the line will be set to the least significant bit of the number.

Digital Line references using variables - Variables or numeric expressions may be used to reference digital lines as either individual line numbers or as line numbers of a designated port#. In either case, the designator expressions must be enclosed within parenthesis.

Examples:	LA34 = 1	'Set line 34 in the main module to one (+5 volts)
	LB3,2 = 0	'Set line 2 of port 3 in module "B" to zero (0 v.)
	IF LA9 = 1 THEN END	'Lines tested just like variables
	VAR1 = VAR2 + LD(PORT,LINE)	'Reference by variables

Analog I/O

Once again, the analog input and output information is handled the same as a variable. The main module's analog range is either Ø-10 volts or +/-5 volts, depending on the setting of the ANALOG command. The range is specified by the command ANALOG (see the Reference Section for more information about the ANALOG command). Expansion Modules may have different ranges and resolutions. Consult the manual accompanying each Expansion Module for information.

All Analog Inputs have a voltage resolution of about 2.44 millivolts. The voltage resolution at Analog Outputs 1-4 is about 2.44 millivolts and about 39.2 millivolts at Outputs 5-8.

When an analog output line is set, the actual voltage on the line will be set to the nearest resolution increment. In other words, the voltage is spread in a fixed number of steps, and will always be equal to the nearest step voltage. This is true of input voltages as well. This is merely a characteristic of analog-digital conversion.

Note: At power-up, reset, or whenever a program is started, the main unit analog I/O range is automatically set to 0-10 volts, and all Analog Outputs are set to Ø volts out. When the analog voltage range is changed by the ANALOG command, all analog outputs are set to Ø volts automatically.

Analog line references begin with the letter, "A", followed by the module designator (A-H), then the analog line number. The main module is always "A". This is true for input as well as output analog lines.

Available Analog Lines in Main Unit:

AA1, AA2, AA3, AA4, AA5, AA6, AA7	Analog Output lines
AA9, AA10, AA11, AA12, AA13, AA14, AA15, AA16	Analog Input lines

Examples:

```
AA2 = 3.4                          'Set analog line# 2 of the main module to 3.4 volts
AC4 = AD12 + AA3                   'Read and set just like variables
IF AB3 < 4.5 AND AE8 > 2.1 THEN END 'Tested like variables
AB(VAR1) = AD(VAR2)                'Analog I/O referenced by variables
```

External Trigger

The FT-100a main unit has one 8-bit digital input port that has an external trigger line. This trigger line may be used to latch the port information coincident with an external trigger event. The polarity of the trigger is programmable, and the trigger can be reset through program commands. External modules may have other externally latched ports as well. See the module manual for information.

If the external trigger capability is not needed, the digital input port may be used like any other input port.

Setting the Trigger Polarity- The reference designator is "TP", followed by the module designator, followed by the Digital Input port number to be triggered. The main module has only one gated port, Port# 10. Setting the trigger polarity to one (1) causes the gate to latch on a low-to-high transition on the trigger line. A setting of zero (Ø) will trigger on a high-to-low transition. (Note: Latching occurs only upon input voltage transitions at the trigger input.)

Available Trigger Polarity Designator in Main Unit: TPA10

Examples:

```
TPA10 = 1          'Main module gated port will trigger on low-to-high transition
TPB7 = VAR1 / 3.4  'If trig pol is set to other than "Ø", it will equal the least
                   'significant bit of the number.
VAR1 = TPD9 +3.5   'Read like a variable
```

Resetting the Latch - The reference designator is "LR", followed by the module designator, followed by the digital input port number. The latches for all gated ports should be reset just before "looking" for a trigger event. This insures the state of the latch. Once reset, the latch should stay "ready" until a trigger event. Caution: External static or voltage transients can cause the latch to trigger unexpectedly.
The polarity of the trigger for a gated port has no effect on resetting the trigger.
Note: Latch Reset is a "set only" command and cannot be read.

An externally triggered input port is also usable for general purpose digital input as long as the latch is reset.

Available Latch Reset Designator in Main Unit: LRA10

Examples:

```
LRA10    'Reset trigger on main module
LRC6     'Reset trigger on gated port# 6 in module "C"
```

Reading the Latch Status - The reference designator is "LS", followed by the module designator, followed by the digital input number. This I/O reference indicates the "status" of the trigger for a gated port. It is a "read-only" reference, and will equal zero (Ø) if latch is not triggered, and one (1) if it is triggered.

Available Latch Status Designator in Main Unit: LSA10

Examples:

```
DISP LSA10                    'Display status of the latch in main module
IF LSC7 = 1 THEN GOTO LABEL1  'Test like a variable
VAR1=LSA10 + VAR2             'May be used in expressions.
```

3.5 Setting the Variable Voltage Supply

The variable voltage supply is a special analog output that can serve as a power supply for the unit connected to the FT-100a. This programmable supply is regulated, can handling up to one amp of current, is overload protected, and can be set to any voltage between 0 and 10 volts with a resolution of about 39.2 mv. The variable supply output may be set and read like a variable, with its own fixed name, SUPPLY. The analog output address AA8 is the same line as SUPPLY and may be used interchangeably. Although this line is capable of high current output, it may also be used as a general purpose analog output.

CAUTION! When a program ends, either by reaching the end of the program or by a STOP command, the variable voltage supply output will <u>remain</u> at the last setting!

Syntax: SUPPLY {voltage}
or: SUPPLY = {voltage}
or: AA8 = {voltage}

{voltage} - Any number or numerical expression that evaluates to 0-10.

Examples:	SUPPLY 4.5	'Set the variable voltage supply to 4.5 v.
	SUPPLY = VAR1	'Set the var volt supply to value of variable VAR1
	VAR1 = SUPPLY*3.4	'Use SUPPLY like a variable

3.6 Loops and Jumps

Normally a program flows from one statement to the following statement, but often it is desirable to have the flow of the program broken and "jump" to a different location. Frequently, we need to repeat a segment of a routine many times within a program. The following commands are used to accomplish these tasks.

GOTO - Causes the program to immediately jump to the indicated label. This is the simplest form of altering the program flow. The referenced label may be anywhere within the program. The program will unconditionally jump to the designated label.

Syntax: GOTO {label}

Example: GOTO LABEL1 'Immediately jump to LABEL1

CALL - Subroutine "call". Similar to GOTO, but each subroutine has a RETURN (return) statement to direct the program back to the location immediately <u>following</u> the CALL statement. Subroutines are simply small programs, within a larger program, that can be used many times. Subroutines provide a very efficient method of repeating given tasks several times without the need to duplicate program lines.

Subroutines may be "nested" within other subroutines. In other words, subroutines may be "called" from within other subroutines.

Syntax: CALL {subroutine name}

{subroutine name} - The name of the called subroutine. This is a label located at the first line of the subroutine. Subroutine name <u>cannot</u> be a string variable.

Example: CALL SUB1 'Immediately jump to SUB1 but remember location of CALL 'statement.

RETURN or RET - Command to instruct the program to return to the location after the original CALL statement. When the original CALL statement was executed, the FT-100a "remembered" the location of the CALL statement. The program will jump back and begin executing of the statement immediately following the original CALL statement. RET is a shortened version of the command.

Syntax: RETURN
or: RET

Example of subroutine CALL and RET:

```
        INC PA5
        CALL SUB1
        INC PA5
        CALL SUB1
        PRINT VAR1
        END
'
SUB1:   VAR1 = VAR1 + PA1
        DISP VAR1,PA1
        RETURN
/
```

In the previous example, the program increments the output port PA5, then jumps to the subroutine SUB1 and adds the input port PA1 value to variable VAR1, then displays the values of VAR1 and PA1. The subroutine then returns to the statement after the CALL command. Port PA5 will then be incremented, and again the program will jump to the same subroutine SUB1 and add PA1 to VAR1 and return. The program will then print the variable VAR1.

FOR/NEXT, LOOP/ELOOP - (Note: FOR and LOOP are exactly the same, as are NEXT and ELOOP. They may be used interchangeably.) These statements are used to "block in" a section of the program and go through the given block of program commands a given number of times. This ability is a very efficient way of performing a task many times with a minimum of programming. The initial FOR (LOOP) statement sets a variable to a given value and defines the maximum value for the variable within the loop. Each time a NEXT (ELOOP) statement is encountered, the variable is incremented by one and compared to the maximum value. If it is less than or equal to the maximum value, the program will "loop" back to the statement just after the FOR (LOOP) statement and go through the loop again. When the loop variable is larger than the maximum value, the program will not "loop" back, but instead continue through to the next statement after the NEXT (ELOOP) statement.

It is not normally a good idea to branch or jump out of a loop since the program doesn't know if you want to be in the loop or not. The loop counter will still be active. The best way to unconditionally jump out of a loop is to set the loop variable to a value equal to or above its maximum value and then jump to the "NEXT" statement. This will "fool" the program into "thinking" the loop is finished. The loop counter will then be eliminated. If a program "ends" while still "inside" a loop, the "FOR w/o NEXT" error will occur.

Loops may be "nested" within other loops up to 8 loops deep.

Examples:

```
FOR VAR1 = 4 TO 12          'Program will loop through routine 9 times,
  VAR2 = VAR2 + VAR1        'adding VAR1 to VAR2 each time
NEXT

FOR X = 2 TO 8              'One loop nested within another
  FOR Y = 0 TO 3
    —
    —
  NEXT
NEXT
```

3.7 Conditional Statements (IF . . . THEN . . .)

The IF/THEN statement has the ability to evaluate a condition and execute commands if the condition is "true". If the comparison expression is evaluated as "true", all commands following THEN, on the same line, will be executed. If the comparison is "false", the program "falls through" the IF/THEN statement and begins execution of the following command. Any expression that evaluates to a number that is non-zero will be declared as "true".

Syntax: IF {comparison expression} THEN {conditional commands}

{comparison expression} - Two or more expressions with comparison operators. Multiple comparison expressions may be joined in the same statement by Boolean operators.
{conditional commands} - Any valid program commands or statements.

The comparison expression, in its simplest form, is one expression compared to another. However, multiple comparisons may be made within the same IF/THEN statement by using Boolean operators. Any of the expressions within the comparison expression may be unresolved as well. String expressions may be compared as well as numerical expressions. See Operator Types for the various comparison operators available.

The conditional commands following THEN may be any valid program commands. If the comparison is "true", the program will execute everything after THEN (on the same line). Since more than one command can be placed on a line (separated by a colon), several commands can follow "THEN", and will be executed if the comparison is "true". If the comparison is "false", everything after THEN is ignored, and the program "falls through" to the line following the IF/THEN statement.

The best way to understand the IF/THEN statement is to look at some examples with explanations.

Example: IF VAR1 = VAR2 THEN BEEP

"VAR1 = VAR2" is the comparison expression and "BEEP" is the conditional command. If the comparison is true, the FT-100a will "beep", otherwise the BEEP command will be ignored and the program will go on to the next statement after the IF/THEN.

Example: IF A > B THEN VAR1=A:VAR2=B

If the variable A is numerically larger than B, the variable VAR1 will be set to the value of A and variable VAR2 set to value of B.

Example: IF (A = B) AND (C = D) AND (E = F) THEN BEEP

Multiple comparisons may be made within the same IF/THEN statement. The entire comparison must be "true" to execute the "THEN" command.

Example: IF ((3.4+VAR1)/256 = VAR2) OR (PA2 < 34H) THEN PA1 = VAR2+VAR1

Expressions may be unresolved in the comparison expression.

Example Routine:

```
      AA1 = 0
BEG:  IF AA9 > 7.2 THEN DISP AA1, AA9:BEEP:END
      IF AA1 <= 10.0 THEN AA1 = AA1 + 0.1:GOTO BEG
      DISP "DONE!": END
```

This routine starts by setting analog output AA1 to Ø volts. It then compares the reading at analog input AA9 to 7.2 volts. If it is not higher, the voltage at AA1 is compared to 10 volts and increased by 0.1 volts if it is below 10 volts. The program then goes back to BEG to compare the values again. This will continue until the voltage at AA9 surpasses 7.2 volts. The program will then display the values of AA1 and AA9,

beep, and end the program.

Example: A$ = "abcdef" : B$ = "def"
IF B$ = RIGHT$ (A$,3) THEN BEEP

If string expressions are compared, they must be *exactly* equal for the comparison to be true.

Consult the Reference Section for more examples of IF/THEN statements.

3.8 Single-Step Operation

The FT-100a may be operated in a "single-step" mode. This is most useful while writing and debugging programs. While the "single-step" mode is enabled, the FT-100a will execute one command or statement and halt (with an audible "beep"). When the CONT pushbutton is pressed, the next command line is executed. The line number is displayed on the LCD screen.

Single-step mode is enabled and disabled through the "single-step command". This command may be issued as a "direct" command or it may be incorporated as a program command. When the program is in a "single-step" wait state, the operator may communicate with the FT-100a via the keyboard or through the serial line, and may read and even alter variables, etc. The "single-step" mode may even be disabled through this method. Selectively enabling and disabling single-step mode *within* a program can make debugging large programs much easier.

Single-step Commands:

SS ON - Turn "on" single-step mode of operation. Single-step will remain "on" until disabled by an "SS OFF" command.

SS OFF - Disable single-step mode of operation. Single-step will remain disabled until turned "on" by another "SS ON" command.

When single-step mode is turned "on", it will remain in effect until turned "off" by the SS OFF command. Starting or stopping a program will not turn off single-step mode. Single-step is initialized in the "off" state at power-up or after pressing the RESET pushbutton.

For more information, see SS ON/OFF in the Reference section

3.9 Saving the Program

Programs may be "saved" on a floppy disk, or they may be downloaded to another computer through the serial communications port.

The format for each of the specific program save commands are covered in detail elsewhere, but a brief introduction is included here:

Saving Program to Floppy Disk - This may be done from the front panel pushbuttons or as a "direct command" (command issued through the keyboard or serial communications port).

□ Front Panel Control - (See the Disk Control through Front Panel Pushbuttons section in the Disk Operation section)

□ Direct Command - (See the Direct Commands section in the Direct Mode Operation section)

Saving Program to a Computer - The program will be sent out the serial communications port and may be saved in either ASCII or "compacted" format. Downloading a program through the serial port must be done by issuing a "direct" command. The "direct" commands are:

LIST - Send ASCII form of program out serial port.

DUMP - Send "compacted" form of program out serial port.

For information on these "direct" commands, see the Direct Commands section in the Direct Mode Operation chapter and also Direct Commands in the Reference Section.

4.0 Disk Drive Operation

Disk drive functions may be accessed directly through the front panel pushbuttons or through Direct Commands initiated by the keyboard or a computer or terminal. Some disk control functions are available via embedded program commands (software). Information on program commands may be found in the Programming Section and the Reference Section. For information on utilizing Direct Commands, see the Direct Mode Operation section.

> *** * WARNING! * ***
> Pressing the RESET pushbutton, or turning off power, during any disk operation, or while an output disk file is "open", can destroy the disk file!

4.1 Handling Floppy Disks

The FT-100a uses 3.5 inch, 1.44 Megabyte floppy disks (DS,HD). The disk format is compatible with MSDOS, Windows, and Windows 95 used in IBM PC and compatible computers. This permits freely interchanging floppy disks between an IBM compatible computer and the FT-100a. Program and data files can easily be transferred between the two.

The storage medium in floppy disks is very delicate and fragile. Fortunately the disk is mounted inside and protected by a plastic case with a window covering the access. Unfortunately, temperature, humidity, and very rough handling can still be detrimental to the well being of the floppy disk. Always handle floppy disks with care and never expose them to temperature extremes or high humidity. Consult the information that accompanies each box of floppy disks for more handling and storage information.

4.2 Inserting and Removing a Floppy Disk

When inserting a floppy disk into the FT-100a disk drive, be sure the disk is oriented correctly. The floppy disk should be inserted such that the metal shutter goes in first and slides left to open the window. The label should be facing up. Press the disk into the disk drive slot until an audible "click" is heard and the release button "pops" out. It should not take much force to insert the floppy disk. If the disk doesn't insert and latch properly, remove the floppy disk, check the orientation, and try again.

To remove a floppy disk from the FT-100a, simply press the disk release pushbutton located on the disk drive. The floppy disk should release and "pop" out. It may now be pulled the rest of the way out.

4.3 Disk Control through Front Panel Pushbuttons

Most disk drive commands are available through the COMMAND pushbutton. Pressing this pushbutton will halt the FT-100a and cause it to display a command menu. (Caution: Disabling the internal interrupts with the DINT command will incapacitate the COMMAND pushbutton).

When COMMAND is pressed, the display will indicate:

```
Command:  LOAD
  START-  to select
   CONT-  for other
          commands
```

Each time the CONT pushbutton is pressed, the display will scroll through the available commands. To access the indicated command, press START. To get out of this menu, press the ESC pushbutton. If a disk operation is attempted with no floppy disk in the drive, an ERROR will be indicated.

Each of the disk commands are shown below with information necessary for operation:

LOAD - Loads programs from a floppy disk (ASCII and "compacted").

Display will show:

```
{program name}
START-  Load Prog
```

The first program file on the floppy disk will be indicated. Only program files will be shown (.PRG or .ASC extensions). Pressing CONT will scroll through all the program files on the floppy disk. When the desired program is shown, pressing START will immediately load the program.

After the program is loaded, the FT-100a will "beep" and the display will show: "READY". The program is now ready and may be run simply by pressing START. Pressing ESC at any time will exit this routine and go back to the Command menu.

DIR - Displays the directory of the floppy disk.

Display will show:

```
{1st Disk File}  #1
{2nd Disk File}
{3rd Disk File}
```

The first three files on the floppy disk are shown. Note that the 1st file on the floppy disk is indicated by the "#1". Pressing CONT will cause the display to scroll up one line and show the next filename on the third line. When the end of the directory is reached, the listing will start over with file #1.

Pressing ESC will exit this routine and go back to the Command menu.

SAVE - Use to save programs on the floppy disk.

Display will show:

```
{program name}.PRG
START-  Save Prog
CONT - Ascii Save
```

The current program name will be shown along with ".PRG" extension. This indicates that the program will be stored in the "compacted" format. Pressing CONT will change the filetype to ".ASC" (ASCII format). If the program is stored in the ASCII format, the floppy disk may be inserted into an IBM compatible computer to be edited using a word processor or text editor,

To save the program, select the desired format and press START. After the program is saved, the FT-100a will "beep", and the display will go to the main power-up screen.

Pressing ESC will exit this routine and go back to the Command menu.

DELETE - Use to delete any file from floppy disk.

Display will show:

```
 {1st filename}
START- To delete
CONT - Next file
```

The first floppy disk filename is shown. If you do not want to delete this file, pressing CONT will show the next file. When the desired file is shown, press START. The display will show:

```
 {Filename}
Press CONTINUE
 to delete file
```

If ESC is pressed at this point, the previous screen will be shown, allowing you to choose another file to delete. Pressing ESC again will exit and go back to the Command menu.

CONT must be pressed to erase the file. This is a safety check to be sure you want to erase the file. Once erased, the file cannot be retrieved!

FORMAT - In order to use a new floppy disk, it must be "formatted". This, in essence, structures the floppy disk so the FT-100a or computer can locate and access all locations within the floppy disk. Once a floppy disk has been formatted, it should never need to be reformatted.

The format structure and floppy disk storage methods used by the FT-100a is the same as that used by MSDOS on the IBM PC computer. This compatibility between the two allows easy file transfer.

> * * **WARNING** * *
> Formatting a floppy disk will erase the entire disk!
> There is no way of recovering erased data!

Display will show:

```
 * FORMAT *
Press CONTINUE
 to format disk
```

If ESC is pressed at this time, the display will go back to the Command menu.

To format the floppy disk, press CONT. This is a safety check to be sure you want to format the floppy disk. When CONT is pressed, the display will show:

```
  * FORMAT *
 Please stand by
Tracks Remaining
```

The FT-100a will begin formatting the floppy disk and count down the number of tracks remaining to be formatted. When complete, the display will go back to the main power-on screen.

4.4 Disk Access through Program Commands

In order to store and read data on a floppy disk, the program must be able to control the disk drive. The program commands used for disk control may also be executed as direct commands.

When large amounts of data need to be collected, or when data is to be permanently saved, storage on a disk file can be very beneficial. Sometimes a very large amount of data must be observed, but all the data does not need to be kept permanently. The data can be stored in a disk file, and after collection, processed by the program, then automatically deleted.

Since the FT-100a disk format is compatible with the IBM PC format, data collected on a disk file may be physically moved to another computer for processing or analysis. This allows collected data to be directly input into a spread sheet or data analysis program.

In order to store and/or retrieve data on a disk, the desired file <u>must</u> be "opened", using the OPEN command. This tells the FT-100a which file to access. Only one file may be "open" at any given time. It is <u>very</u> important that files be "closed" (using the CLOSE command) after the program is finished accessing the file.

All filenames <u>must</u> be in quotes. String variables may be used to contain the filename. String variables should <u>not</u> be in quotes. The program commands used for disk control are covered in detail in the <u>Reference Section</u>, but will be briefly described here as well:

OPEN - Used to "open" a data file in order for the program to *read from* or *write to* the opened file.

Syntax: OPEN {filename},{filetype}{,APPEND}

{filename} - Name of the file to be accessed.
{filetype} - "IN" for "input" file
"OUT" for "output" file.
{,APPEND} - An optional parameter used only with output files to designate that the new data be "appended" to the end of the old data.

(Note: The command words: IN, OUT, and APPEND may be abbreviated to only the first letter or each word.)

CLOSE - Will "close" any file that is open (Input or Output).

Syntax: CLOSE {filename}

{filename} - Optional. Only used to clarify program flow. In this command, the filename does not need to be in quotes.

DEL - Deletes any specified file on the floppy disk.

Syntax: DEL {filename}

{filename} - Name of the file to be deleted, and must be in quotes. The filename may be a string variable.

Note: The file will be deleted immediately, with no second "check".

STORE - Stores data in the open disk file.

Syntax: STORE {data},{data}, ...

{data} is one piece of data to be stored in the disk file. Data may be a number, string, a variable, or an expression. Multiple data may be stored with one STORE command if separated by commas.

Note: All disk file data is stored as its ASCII string equivalent.

INPUT - Read data from the open disk file.

Syntax: INPUT {variable},{variable}, ...

{variable} is any variable that is compatible with the data being read in form the disk. Each piece of data is read in sequentially and placed in the designated variable. Multiple data may be input with one INPUT command, separated by commas.

Note: For more information about disk storage, see the Data Storage Section.

Example of a program that collects and stores data in a disk file:

```
DATA_GET                'Program name
'
OPEN "FILE1",OUT        'Open the output file "FILE1"
FOR X = 1 TO 500        'Loop through 500 times
STORE AA4, TIME$        'Collect data from analog input AA4 and store, with time of day,
                        'in the disk file
DELAY 60000             'Delay 1 minute
NEXT
CLOSE FILE1             'Close the file
/
```

This program collects 1000 samples of the analog input line AA4 and stores the data and the time of day on the disk in the file, "FILE1".

The following program segment will read in the data from the disk and place the data in an array:

```
DATA_USE                'Program name
'
DIM A(2,500)            'Dimension the array
OPEN "FILE1",IN         'Open the input file "FILE1"
FOR X = 1 TO 500        'Loop through 500 times
INPUT A(X)              'Read data from the disk and place in array.
NEXT
CLOSE FILE1             'Close the file
/
```

Note: For more information about disk storage, see the Data Storage Section.

Notes: 1. Use Microsoft Word as the program editor. Then save to floppy the file with the extension: MS-DOS Text with Layout. This will save the program in ASCII format that the FT-100a can read.

5.0 Direct Mode Operation

The FT-100a has the ability to execute commands in a "direct mode". In other words, when a command is sent to the FT-100a through the optional keyboard or serial communications port, it will be executed immediately. This can be very beneficial when manually controlling a device, "ringing out" the hardware interface, or when troubleshooting a defective unit connected to the FT-100a.

Nearly all programming commands and statements are executable in the direct mode. The syntax is the same and the operation is normally the same. Following are the program statements that are not available in the direct mode:

CALL, RET, DINT, EINT, READ, DATA.

All other programming commands and statements are directly executable in the direct mode. The syntax and operation of the programming commands may be found in the Reference Section.

While a program is running, it is sometimes possible to communicate with the FT-100a through the keyboard or serial line. Because some commands and statements "shut down" the serial port for brief periods of time, reliable serial communications is not always possible. The keyboard input as well as the serial communications input use internal interrupts for proper operation. If the test program disables the interrupts (using the DINT command), neither method of input can be used until the interrupts are enabled (using the EINT command).

In addition to the programming commands, there are some commands that are available as "direct" commands only. These commands are available through the keyboard input or serial port but are **not** valid as programming commands. Below is a list of the "direct commands" and a description of each. More information may be found in the Reference Section.

5.1 Direct Commands

The syntax for a direct command is normally the keyword followed by a carriage return. Any exceptions will be noted below:

START - Immediately starts running the program in the FT-100a. (Same function as START pushbutton)

RUN - Same as START

STOP - Causes the program to immediately stop. (Same function as the STOP pushbutton)

WAIT - If a program is running, will cause it to pause. (Same function as the WAIT pushbutton)

CONT - If the program has been "paused" by a WAIT command (program or direct), CONT will cause the program to begin with the next statement. (Same function as CONT pushbutton)

INFO - This command causes the FT-100a to send various information out the serial communications line, including the current version of firmware, amount of program memory used thus far, remaining memory etc.

DIR - A listing of the floppy disk directory is sent out the serial communications port.

FILES - Same command as DIR.

RESET - This is a "soft" reset command. The RESET command will cause the FT-100a to go into its "boot-up" routine. This is the same routine that the FT-100a goes through each time power is first turned on. RESET will <u>not</u> erase the program memory, but will erase any data that may be in the FT-100a that hasn't been saved to a floppy disk. This command is similar to pressing the rear panel RESET pushbutton, however there are some internal hardware devices that can only be reset through a "hard" reset.

> * * CAUTION * *
> <u>Never</u> issue a RESET command with an open disk output file!
> The disk file will be destroyed!

LIST - Causes the FT-100a to send an ASCII listing of the program out the serial communications port. The program will be listed in the ASCII format, each line numbered, allowing editing by a word processor or text editor program.

DUMP - Causes the FT-100a to send a listing of the "compacted" program out the serial communications port. The listing is an ASCII representation of the hexadecimal value for each byte of program. There is some header information sent along with the program. This format may be loaded back into the FT-100a through the serial port. The compacted form is the most efficient form of program transfer and is recommended whenever possible.

SAVE - Saves the current program to the floppy disk. The program may be saved in the ASCII format or as a compacted program.

Syntax: SAVE (Save in compacted format)

SAVE,A (Save in ASCII format)

Program will be immediately saved to the floppy disk. The filename will be the same as the Program Name. plus the proper extension. If the ",A" is placed after SAVE, the program will be saved in the ASCII text format, thus allowing reading or editing on an IBM compatible computer.

LOAD - Load a program into the FT-100a. This command is used to load programs from the floppy disk as well as through the serial communications port. The syntax determines the source. Programs transferred through the serial port have identifying headers, so the FT-100a will know the proper format.

LOAD {program name} Loads a program from the floppy disk.

{program name} must be in quotes. If the extension is omitted, the FT-100a assumes ".PRG".

LOAD Load a program through the serial communications port.

Programs loaded through the serial port may be in either the ASCII format or the "compacted" format. When the command is executed, the FT-100a will indicate it is waiting for the program file. The program file has a header to identify the program format. After the program is loaded, the FT-100a will "beep" and indicate "Ready". The program may now be run.

FORMAT - Performs a complete format of the floppy disk. This command causes the FT-100a to begin the "format" routine as described in the <u>Disk Operation Section</u>.

6.0 Data Storage

The FT-100a is capable of acquiring and storing large amounts of information. Data may be handled in several ways depending on its ultimate use and storage time. Since data is sometimes needed for short term and long term purposes, more than one data handling method is often used.

6.1 Temporary Data

If data is not to be permanently saved, and the volume is not too great, it may be simply saved in variables or arrays. When the program is finished, the data will be destroyed. This is the simplest method of temporarily saving data.

If data is to be observed but not saved for later use, the data may be displayed on the front panel display or sent out the serial communications port for display on a computer or terminal. This method provides a very good method of monitoring an ongoing event in real-time.

6.2 Printing Data

One of the most permanent methods of saving data is to print the information on a serial printer. Connection of the printer is covered in the Serial Communications section under Printer Output. Once the printer is connected, and the communications parameters have been set, all data sent out the serial port will be printed.

The PRINT command sends data out the serial port. Data may be numerical, character strings, or combinations of the two.

Note: If a serial port expander option has been connected to the FT-100a, several devices may be addressed through the serial communications port.

See the PRINT command in the Reference Section for more information.

6.3 Sending Data to a Computer or Terminal

Instead of sending data to a printer through the serial communications port, it could be sent to a computer or terminal. The communicating device could then either view the data or permanently store the information on a floppy disk, hard disk drive, tape drive, or any other method. Unlimited amounts of information may be collected and stored this way. Storage capabilities are only limited by the computer or terminal storage capacities.

If a computer, running a communications program, is used as the communicating device, consult the manual for the communications program for information on receiving and saving data. If the data is to be saved to a disk drive, most programs have a "capture" mode, automatically sending all incoming information to the disk file.

Even though serial data is sent to a computer or terminal, the PRINT command is still used. Remember, the PRINT command always sends data out the serial port. The FT-100a doesn't know about its destination or use.

There are many commercial data collection and evaluation programs available for computers. Any of these programs will work with the FT-100a as long as they support RS-232 serial input. The manual with each program will cover set-up and use of the program. Consult Serial Communications in the Reference Section and the Appendix for information on the FT-100a serial interface. Most commercial data loggers should interface as well.

6.4 Storing Data on a Floppy Disk

Data may be permanently stored on a floppy disk in the FT-100a. This is one of the easiest methods of storing large amounts of data. The stored data may be accessed and processed later by an FT-100a program, or the disk may be physically moved to a PC compatible computer for processing. The floppy disk is capable of storing very large amounts of data.

Storing data on a disk in the FT-100a is very simple. The first step is to "open" an output file. This process "tells" the FT-100a in which file to store the data. After the file is opened, each STORE command will place data in sequential order in the opened file. After all data has been stored on the disk, "close" the file with the CLOSE command. Detailed information about opening and closing disk files may be found in the Disk Operation Section as well as under individual disk commands in the Reference Section.

Opening the File

In order to store data on the floppy disk, a file must be "opened" as an "output" file. To "open" an output file, use the OPEN command:

OPEN {filename},OUT

{filename} - Name of output file to be opened. Must be no more than 8 characters plus decimal point (.) and 3 additional "extension" characters (optional), and must begin with a character A-Z. If filename is not a string expression, the filename must be in quotes.

The named file will be opened as the current data file. If the file already exists, the old file contents will be destroyed and the new data will overwrite. Each time data is "stored" to disk it will go into this file as long as the file remains open. The file should be closed (using the CLOSE command) when all data has been stored.

OPEN {filename},OUT,APPEND

Same command as above, except the extra parameter, ",APPEND" (or abbreviated as ",A") tells the FT-100a to "append" all new data to any data that is currently in the file. This will preserve the existing data, adding the new data to the end of the file.

Note: For the OPEN command, "OUT" and "APPEND" may be abbreviated to only the first letter ("O" & "A").

Saving the Data

After the file has been opened, all data stored on the floppy disk will be placed in the "opened" file. To store data, use the STORE command:

STORE {data},{data},·····

{data} - May be numerical or string data.

The data will be stored in the "open" file in sequential order and in ASCII format. Numerical and string data may be intermixed in the same file and even the same STORE command. As many pieces of data as fits on one line (128 characters) can be stored with one command. Multiple data must be separated by commas.

Please note that all data is stored *sequentially*. When data is read back, the sequence will be the same. Be careful that the sequence is known so the data can be processed properly!

Closing the File

After all data has been saved to the opened file, the file must be "closed". This will prevent any inadvertent erasure of the file contents and assure that all the data is stored in the disk file.

Data files are closed using the CLOSE command:

CLOSE {filename}

{filename} is optional and is ignored by the FT-100a. Consequently, filename does not need to be in quotes, nor even match the actual open filename.

The CLOSE command is used to close "input" as well as "output" files.

***** WARNING! *****
If an output file is "open" and either: a RESET command is issued, the RESET pushbutton pressed, or Power turned off, the disk file may be destroyed!
Always CLOSE files when finished!

Notes about Disk Storage of Data

- Only one disk file may be "open" at any given time.
- All information is stored on the floppy disk as ASCII characters.
- If a *number* is input from a disk *into a number variable*, it will be automatically converted back into the number format. If, however, a *number* is input into a *string variable*, the string variable will contain the ASCII representation of the number. If a *character string* is input into a *number variable*, it will also be converted into number form as long as it is an ASCII representation of a number.
- All data is stored on the disk in sequential order. Consequentially, the data must be input in the same order in which it was stored.
- Each stored data element is followed by Carriage Return (0DH) and Line Feed (0AH) characters as a delimiter. When viewed on a word processor or text editor, each data element will be on a line by itself. If data must be stored in a given format for viewing (like a table or chart), use the string function statements to organize and format the data into the desired structure prior to storing on disk.
- If the FT-100a is to print or display stored data in a given format (table or chart for instance), it is usually better to *store* the data as individual data elements. When the data is read back in from the disk, it may be formatted or structured as needed by the print command. This keeps the data separated and accessible for processing.

7.0 Expansion Modules

The FT-100a can expand its input and output capabilities as well as add many other features through external Expansion Modules. These are optional accessories that connect to the FT-100a through a rear panel flat cable connector. Up to seven additional modules may be attached to the base unit.

Because of the vast differences between the various expansion modules available, the interface and operation of each is covered in the manual that accompanies each module. Before connecting the module to the FT-100a, read the manual for the module. Hardware interfacing and programming information will be clearly explained.

I/O Addressing

If an expansion module adds extra input and output ports or lines, program addressing is done in a conventional and predictable way. Each module has a module designator. This is a letter from A to H, with the FT-100a base unit always being module "A". Each of the other modules can be programmed through convenient "DIP" switches or jumpers to any other module designator.

> *** * * WARNING * * ***
> Never program more than one module with a given module designator! Damage to the FT-100a and module could result!

(Programming the module designator is covered in the module's manual).

All I/O references consist of the following:

{I/O type} {module designator} {reference designator}

{I/O type} - P = Digital Port
L = Digital Line
A = Analog Line
TP = Trigger Polarity
LS = Latch Status
LR = Latch Reset

{module designator} - "A", "B", "C", "D", "E", "F", "G", "H" (main unit is always "A")

{reference designator} - Number assigned sequentially to individual I/O.

More I/O reference information may be found in the Reference Section.

The manual accompanying each external module will describe all the input and output lines and ports available, along with the addressing information for each.

8.0 Reference Section

The reference section is provided as a method of quickly finding information about each command or statement. Program commands and Direct commands are separated for ease of use. When a working understanding of the FT-100a is achieved, most programming as well as interfacing can usually be accomplished by seeking specific information in the Reference Section or the Appendices.

If you are unable to find the needed information in the Reference Section or the Appendices, go back through the manual and study the area in question.

8.1 Reference - Program Commands

AA, AB, AC, AD, AE, AF, AG, AH

Type: Analog Input/Output Line Reference Designators

Purpose: Directly address analog input and output lines as variables.

Syntax: A{module}{line number}

{module} - Test module (A-H). The FT-100a main unit is always module A.

{line number} - The individual line number for the analog input or output line. May be a numeric expression within parenthesis.

Notes: Analog input and output lines are addressed like any other variable. The second letter in the designator indicates the module designator where the analog line is located. Module designators are set by "DIP" switches or jumpers, and are unique for each separate module.

The line number is the individual analog line number. These numbers start at 1 and progress upward for as many analog lines there are in the module. For information on the analog line references for the main unit, look in the Appendix under Hardware Interfacing. Also see the Programming Section under Addressing I/O.. Reference information for external modules will be in the manual for each individual module.

Examples:

```
AA3=3.4            'Set analog output line 3 in the main module to 3.4 volts.
VAR1=AB4*6.8       'Read analog input line 4 in module "B", multiply by 6.8, then
                   'put the result in variable VAR1
AD(VAR1)=AA3       'Set analog output, referenced by variable VAR1 in module "D",
                   'equal to analog output line 3 in 'the main module.
```

ABS

Type: Numerical Function

Purpose: Find and return the absolute value of a numerical expression. (Result is always positive)

Syntax: ABS ({numerical expression})

{numerical expression} - Any expression that can be evaluated to yeild a numerical result.

Examples:

```
X = ABS (4.5-8.3)        'Set variable X to 3.8
DISP ABS (VAR1 -26.2)    'Display absolute value of VAR1-26.2
```

ANALOG

Type: Program Command

Purpose: Change Analog Input/Output voltage range in main unit

Syntax: ANALOG {voltage}

{voltage} - Highest voltage in desired analog voltage range.
10 (ten) for 0-10 volt range
5 (five) for +/-5 volt range
(Voltage may be a numerical expression)

Notes: The FT-100a main unit has the added feature of a selectable analog input/output range. The ANALOG command is used to select the range. Any time the range is changed, all analog output voltages are automatically set to 0 volts, regardless of the range.

At power-up, reset, or whenever a program is started, the analog I/O range is automatically set to 0-10 volts, and all Analog Outputs are set to Ø volts out. Stopping or ending a program will not change the Analog Range.

Changing the Analog Voltage Range has no effect on Expansion Modules.

Example:

VOLTAGE 5	'Set the analog I/O range to +/- 5 volts.
VOLTAGE 10	'Set the analog I/O range to ~~+/- 5 volts.~~ 0 - 10 volts.
VOLTAGE (VAR1 + VAR2)	'voltage may be a numerical expression.

AND

Type: Logical operator

Purpose: Used in Boolean operations to compare bits. Will cause the logical AND of two numerical expressions.

Syntax: {numerical expression} AND {numerical expression}

Examples:

VAR1 = 5	
VAR2 = 14	
X = VAR1 AND VAR2	'Set variable X to 4 [5 AND 14]
DISP (VAR1 OR 16) AND VAR2	'Display 4 [(5 OR 16) AND 14]

AND can be used in IF/THEN statements to combine comparisons:

IF ((A=(B+3)) AND A$="ABC") OR PA3=5CH THEN GOTO LABEL1

(See Operator Types in the Programming Section for more information about Logical Operators)

ASC

Type: Numerical function

Purpose: Find and return the ASCII code for a string expression. Result is a number.

Syntax: ASC ({string expression})

{string expression} - Any expression that, when evaluated, results in a string.

Notes: If {string expression} evaluates to a multi-character string, ASC will return the ASCII code for the first character in the evaluated string.

Examples:

```
VAR1 = ASC ("ABC" + "DEF")   'Set variable VAR1 to 65 (ASCII code for "A")
VAR2$ = "dog"                'Set string variable VAR2$ to "dog"
DISP ASC (VAR2$)             'Display 100 (ASCII code for "d")
```

BEEP

Type: Program Command

Purpose: Produce tone in sound enunciator

Syntax: BEEP {frequency},{duration}

{frequency} - Frequency of tone in Hz. (1-65535) Default ≈ 1500 Hz

{duration} - Duration of tone in milliseconds (1-65535) Default ≈ 250 ms.

Notes: {frequency} and {duration} may be numerical expressions, and both may be optional. If either parameter is omitted, the default value will be used. If {duration} is omitted, the delimiter (comma) may be omitted as well, but if {frequency} is omitted and {duration} is specified, the delimiter (comma) must be present.

Examples:

```
BEEP 1000,2500               '1000 Hz tone for 2.5 seconds
BEEP 2400                    '2400 Hz tone for 250 milliseconds
VAR1 = 500
BEEP 1000+1500,VAR1          '2500 Hz tone for 500 milliseconds
BEEP ,1000                   '1500 Hz tone for 1 second
```

CALL

Type: Program Command

Purpose: Access subroutines.

Syntax: CALL {subroutine name}

{subroutine name} - Label attached to the desired subroutine.

Notes: This command causes the program to jump to the location of the specified subroutine. The subroutine should have a return statement (RETURN) to instruct the program to jump back to the statement just after the CALL command.

{subroutine name} must be a valid label, and can be located anywhere within the program. Subroutines may be "called" from inside other subroutines.

Examples:

```
CALL SUB1            'Jump to subroutine SUB1
(next statement)     'When the program reaches a RETURN (RET) command in the
                     'subroutine, it will return to this statement.
```

CHR$

Type: String Function

Purpose: Convert a numerical ASCII code into its equivalent string.

Syntax: CHR$ ({numerical expression})

Notes: {numerical expression} must evaluate to a value of 1-255.

Examples:

```
DISP CHR$ (65)              'Display "A" (65 = ASCII code for A)
A$ = "Na"+CHR$ (109)+"e"    'Set string variable A$ to "Name"
```

CLOSE

Type: Disk File Command

Purpose: Close any disk data file that may be open (Input or Output)

Syntax: CLOSE {filename}

{filename} - (optional) Name of file to be closed. Inclusion of the filename is optional but may help in reading and following the program. Filename does not need to be in quotes.

Notes: If a disk file has been "opened", it must be "closed" to prevent inadvertently destroying or altering the file. If an output file is left open and power turned off (or RESET command issued), part or all of the data file will be destroyed!

It is always considered good programming practice to close all disk data files upon completion.

Under the following conditions, files are automatically closed:
- Program started
- STOP pushbutton pressed
- STOP command issued (direct command)
- ERROR condition
- End of program reached

Inclusion of the filename with the CLOSE command is optional and is ignored by the FT-100a. If an incorrect filename is used, an error will not be indicated, and any open file will be closed properly.

For more information, see OPEN in Reference Section.

Examples:

```
OPEN "FILE1",OUT      'Open output data file FILE1
STORE VAR1            'Store value in variable VAR1 to file FILE1
CLOSE                 'Close FILE1
```

CLR

Type: Program Command

Purpose: Erase and delete all variables (including arrays)

Syntax: CLR

Notes: All variables are deleted. This will free up any memory used by previous variables. CLR has no affect on I/O lines.

Care should be taken if the CLR command is inside a FOR/NEXT loop, since a variable is used to track the loop. All variables will be eliminated and assume values of zero (Ø).

Example:

```
VAR1 = 4.5            'Set VAR1 to value of 4.5
A(3,4) = 3+7.2        'Set array element a(3,4) to 10.2
CLR                   'Completely erase all variables and array elements
```

CLS

Type: Program Command

Purpose: Clear characters from LCD display.

Syntax: CLS {line number}

{line number} - (optional) LCD display line number to be cleared.

Notes: {line number} must be an integer between 1 and 4, and cannot be an expression. If {line number} is omitted, or is anything other than an integer between 1 and 4, the entire screen is cleared. If only one line is cleared, the cursor will be placed at the leftmost space of the cleared line. If the entire screen is cleared, the cursor will be located at the top left position.

Example:

```
CLS          'Clear the entire LCD display
CLS 3        'Clear row number 3 of LCD display
CLS VAR1     'ERROR! Line number must be an integer
```

COM

Type: Program Command

Purpose: Change Serial Communications protocol

Syntax: COM {baud},{Auto LF},{# of Columns},{Number Format},{Echo},{Char Delay},{Direct Mode}

{baud} - Baud rate for serial communications (110, 300, 600, 1200, 2400, 4800. 9600)

{Auto LF} - Ø (zero) if automatic linefeed is not desired with every carriage return.
1 (one) if automatic linefeed is to be sent with every carriage return.

{# of Columns} - Number of printer columns (40, 80, 120, X=Unlimited)(Note 1)

{Number Format} - Format in which numbers will be displayed and printed
D = Decimal (4.57, 0.235, 156)
H = Hexadecimal (3B4CH, 45H, AF3H)
B = Binary (10110001B, 101B, 11011B)
E = Exponential (3.4E12, 8.223E-21)
D8 = 8-bit Decimal (note 2)
H8 = 8-bit Hexadecimal (note 2)
B8 = 8-bit Binary (note 2)
E8 = 8-bit Exponential (note 2)

{Echo} - Ø (zero) to disable Echo (note 3)
1 (one) to enable Echo (note 3)

{Char Delay} - (1-32000) Relative delay that is inserted after each serial character sent out the serial communications line. (note 4)

{Direct Mode} - Ø (zero) to disable Direct Commands (note 5)
1 (one) to enable Direct Commands (note 5)

Note 1 - "# of Columns" should normally be set to "Unlimited" unless you are using a printer that does not issue an automatic carriage return at the end of each line. Some data terminals may be set up this way as well.

Note 2. "8-bit" data format will cause all numbers to be displayed and printed as truncated to the 8 least significant binary bits. This is very useful when working with 8 or 16 bit digital information. CAUTION: When a number is displayed in 8-bit format, there is no way of knowing what, if any, bits were truncated!

Note 3. Enabling Echo will normally cause all characters sent to the FT-100a through the serial port to be "echoed" back through the serial line. The only exception to this is during program loading.

Note 4. This delay permits higher baud rate communications (no handshaking) with devices that may not be fast enough to process the incoming data. Experimentation may be necessary if you are having trouble communicating with a device. Default value is set to 357, and is set only during a System Reset. Issuing a RESET command, or pressing the Reset pushbutton, will not change the character delay value.

Note 5. When no program is running, the FT-100a will interpret all incoming serial character strings as "direct" commands. This is also true while a program is running if the Direct Command Mode has not been disabled. The Direct Command Mode can be disabled only during program execution. When the Direct Command Mode is disabled, all incoming serial data is placed in the serial input buffer and may be read through the GET command. For more information, see the GET command in the Reference section.

Notes: There are seven fields in this command, each separated by a comma. If any field is not to be changed, simply omit the information in that field. The field delimiter (comma), following the omitted field, must be present only if there are other fields following. Any field that is not changed will remain at its previous setting. With the exception of the Number Format, all of the parameters within the fields may be numerical expressions.

Each of the parameters set by the COM command, with the exception of the Char. Delay, are temporarily saved when a program begins running. When the program ends, the old values are re-instated. This is also true during a "wait" period (WAIT pushbutton pressed, or WAIT command). When a CONT command is issued, the previous values are again re-instated as the program continues. Any parameters set by the COM command as a direct command, will remain as the default values.

Example:

```
COM 4800,1,80    'Baud=4800, Auto LF after CR, 80 column print
COM ,,,,0,,0     'Echo disabled and Direct Command Mode disabled.
COM ,,X,H8       'Unlimited number of printer columns, Number format set to 8-bit hex.
COM ,,40,B,0     '40 column print, Number format is Binary, Echo is turned off.
COM ,,,,,500     'Serial TX delay set to 500
```

COS

Type: Numeric Function

Purpose: Calculate and return the trigonometric cosine of a number.

Syntax: COS ({numerical expression})

{numerical expression} must be in radians.

Notes: As with all functions, single entity parameters do not require parenthesis.
To convert degrees to radians, use: RADIANS = DEGREES / 57.295778

Examples:

```
DISP COS (2.15)             'Display -0.547358
VAR1 = 34.5 * COS (VAR2)    'Multiply 34.5 by the value of variable VAR2 and place
                            'the result in variable VAR1.
```

CURSOR

Type: Program Command

Purpose: Place cursor in a given location on LCD display.

Syntax: CURSOR {line number},{column number}

{line number} - Vertical line for cursor destination (1-4)

{column number} - Horizontal line for cursor destination (1-16)

Notes: {column number} is optional. If not specified, the default value of one (1) is assumed. If {column number} is omitted, the delimiter (comma) must be omitted. Either or both fields may be numerical expressions.

Examples:

```
CURSOR 3,8        'Move LCD cursor to row 3, column 8
CURSOR 2,VAR1     'Set cursor to row 2, line# specified by 'variable VAR1
CURSOR 4          'Move LCD cursor to row 4, column 1
```

DATA

Type: Statement

Purpose: Permit the entry of several numerical and/or string values in sequence.

Syntax: DATA {data element},{data element},·····

{data element} - May be a number or a character string.
Data elements may be unresolved expressions.

Notes: Data elements within the DATA statement are sequentially read into variables each time a READ command is executed. Once a data element has been read in, it cannot be read in again unless a CLR command is executed. Placement of the DATA statements within the program is unimportant. The first DATA statement in the program is accessed first, and so on. Numerical and string data may be intermixed within the same DATA statement. If this is done, it is important that the variables being "read into" are of the correct type (numerical or string).

If unresolved expressions are used as data elements, it is very important where, and when they are read in. Any variables in the expressions may have different or unknown values at different places in the program.

For more information, see READ in Reference Section.

Examples:

```
DATA 3.4,29.5E-4,1000.003,6
DATA "House",1500,"Car",3.14,"Bicycle",125     'Strings & numbers
DATA VAR1+VAR2,PA4                             'Unresolved expressions
```

DATE$

Type: Dedicated String Variable

Purpose: Set or read the current date as determined by the realtime clock/calendar.

Syntax: DATE$

Notes: DATE$ is a dedicated string variable of the form: "mm-dd-yy"
mm - Month
dd - Day
yy - Year
DATE$ may be set by using the given form, or it may be read. When setting DATE$, the exact form must be used. No extra spaces or blanks may be inserted or omitted. Any attempt to set DATE$ in any other form will result in an error. Once set, the date will remain accurate even after power is removed.

Examples:

```
DISP DATE$              'Display the current date
A$ = DATE$              'Set string variable A$ to current date
DATE$ = "09-27-95"      'Set the date in the realtime clock/calendar
```

DEC

Type:	Numerical Command
Purpose:	Subtract one (1) from an entity
Syntax:	DEC {number variable, numerical array, or digital output port}
Notes:	This command is merely a simple way of subtracting one from an entity. The equate command can accomplish the same objective. If a digital output port is referenced and the value was zero (Ø), the new value will be 255 (0FFH).

Examples:

```
VAR1 = 4.5       'Set variable VAR1 to 4.5
DEC VAR1         'Now VAR1 = 3.5
DEC PC5          'Decrement value at digital output port #5 in module "C".
```

DEL

Type:	Disk File Command
Purpose:	Delete a file from a floppy disk
Syntax:	DEL {filename},{filename},.....

{filename} - Name of any file that is on the disk. If a filename is referenced that is not on the disk, an error will result. Filename can be a string expression. If filename is not a string expression, it <u>must</u> be in quotes.

Notes: When this command is encountered, the referenced file will be immediately erased from the disk, with no request for confirmation. Within a program, this command can be useful when a disk file is used to temporarily store a large amount of data for processing, but the data does not need to be retained.
More than one file can be deleted within one DEL command. Multiple filenames must be separated with a comma (,) delimiter.

Examples:

```
DEL "FILE1"                   'File FILE1 is erased
DEL "FILE2","FILE3",DAT$      'All three files are erased
```

DELAY

Type: Program Command

Purpose: Pause program for a programmable amount of time

Syntax: DELAY {time}

{time} - Delay time in milliseconds. Time may be a numerical expression.

Notes: During execution of the DELAY command, the FT-100a effectively waits for the programmed time to pass, then the next instruction is executed. The delay time is not of high precision and should not be used for critical timing operations. If interrupts are not disabled before execution of the DELAY command, an interrupt can occur during the delay, causing the delay to be longer than programmed.

Sources of interrupts:
- START/STOP pushbutton
- WAIT/CONTINUE pushbutton
- COMMAND pushbutton
- Incoming serial communications

Examples:

```
DELAY 1000      'Delay 1 second
DELAY 35        'Delay 35 milliseconds
DELAY 60000     'Delay 1 minute
DELAY 3.6E6     'Delay 1 hour
```

DIM

Type: Statement

Purpose: Set aside memory for storage of arrays

Syntax: DIM {array with dimensions},{array with dimensions},·····

{array with dimensions} - Array name with the maximum number of elements for each dimension of the array.
Examples: A(3,26), ARR(100,34,2)

Notes: All arrays have a default maximum dimension of 10 elements per dimension. Extra memory <u>must</u> be set aside for arrays with any dimension over 10 elements.

Considerable memory space is used whenever an array is dimensioned, so try to dimension arrays as close as possible to the actual size to be used.

The first time an undimensioned array is encountered, it will automatically be dimensioned to ten (10) or the referenced value, whichever is larger. This is fine as long as the reference is the largest dimension needed.

Examples:

```
DIM ARR1(34,4)
DIM ABC(2,21,4)
```

DINT

Type: System Command

Purpose: Disable microprocessor interrupts

Syntax: DINT

Notes: Interrupts normally override all other functions of a microprocessor and are general caused by an external event over which the programmer may not have control. The FT-100a has several interrupts, and depending on the type of operation the program is performing, it may be advisable to disable the interrupts during a critical part of the operation. The DINT command enables the programmer to disable all microprocessor interrupts. The interrupts will remain disabled until enabled by an EINT command. If DINT is encountered and the interrupts are already disabled, there will be no change.

When power is initially turned on, START/STOP pressed, or RESET initiated either by the front panel pushbutton or external RESET line pulled low, all interrupts are immediately enabled.

It is important to remember that while the interrupts are disabled, the FT-100a cannot receive any serial communications. If commands are being given in realtime through the serial port, and DINT is sent, no more commands can be sent. Data can be sent out the serial port, however.

FT-100a Interrupts affected by DINT and EINT:

- START/STOP pushbutton
- START/STOP remote function (I/O interface)
- WAIT/CONT pushbutton
- WAIT/CONT remote function (I/O interface)
- COMMAND pushbutton
- RS232 serial input (but not output)
- Some external module functions (optional)

Examples:

```
DINT                        'All interrupts are disabled
LA4=1:DELAY 100:LA1=0       'Set LA4 to 1 for 100 ms.
EINT                        'Re-enable interrupts
```

DISP

Type: Program Command

Purpose: Display numerical or string expressions on LCD

Syntax: DISP {format;}{expression}{delimiter}{expression}{delimiter} .....

{format;} - (optional) Changes display format.
In order to change the display format, "#" followed by the format command characters must be inserted in the display statement. More than one format command may be issued, each separated by a comma. The format command may be used at any location within the DISP statement. The format will be changed beginning with the location of the format command and continue until the next format command or the end of the DISP statement. Any format change is only valid for the duration of the given DISP command; after which, the default values are reinstated.

A semicolon (;) <u>must</u> follow the format command.

Format command characters:

XX.XX - Indicates location of decimal point and number of digits to display. Effective only when using the "decimal" number format.
D - Decimal number format
H - Hexadecimal number format
E - Exponential number format
B - Binary number format
D8 - Show 8 least significant bits in decimal format
H8 - Show 8 least significant bits in hexadecimal format
B8 - Show 8 least significant bits in binary format
E8 - Show 8 least significant bits in exponential format
\+ - Place sign (+/-) in front of all numbers.

Note: See COM in <u>Reference Section</u> for more information about number formats.

{expression} - Any valid numerical or string expression

{delimiter} - Data separator. Type of delimiter determines spacing between expressions on the LCD display.

Delimiter Types:
Comma - Leaves <u>one</u> character space between displayed expressions.
Semicolon - <u>No</u> space between expressions.

(Delimiters may also be used at the end of the DISP command)

Notes: The binary number format has a limit of 32 bits. If a number is larger than that, the number will be displayed in hexadecimal format.

In binary and hexadecimal formats, if the number is negative, the display will indicate ØB! or ØH!. If the number has a fractional amount, the fraction will be dropped and the display will have an exclamation point after the number, $58.3_{(10)}$ = 3AH! = 111010B!

If the DISP command has no parameters, the cursor will move to the beginning of the next row on the LCD display.

For information on the results of the various functional operators within the DISP command, look under each function in the reference section. (Functional operators: ABS, TAB, SPC, CHR$, etc.)

Examples:

DISP "Answer: ";VAR1	'Displays:	Answer: 163.45	(VAR1=163.45)
DISP "BIG ";A$;"!"		BIG House!	(A$="House")
DISP TAB (5);82.3		82.3	
DISP "abc";SPC (4);"def"		abc def	
DISP DATE$		09-27-95	
DISP VAR1,#H;VAR1		163.45 A3H!	
DISP (6+VAR1)/2		84.725	

END

Type: Statement

Purpose: Stops program execution

Syntax: END

Notes: The END statement is normally used to mark the end of a program. It is considered good programming practice to use the END at the end of all programs. This prevents the FT-100a from inadvertently "passing through" the end of program. When a program is loaded in through the serial port, the end of the program is marked internally. As long as this marker is not disturbed, the end of the program will be valid. The END statement can be considered insurance against unforeseen problems. If a program does not have an END statement, and has subroutines following the main program body, it will pass through into the subroutines, giving an error when the RETURN is reached.

When the program reaches an END statement and stops, the only way to restart the program is to press the START pushbutton or issue a START command through the serial input.

Examples: END 'Program stops here

EINT

Type: System Command

Purpose: Enable microprocessor interrupts

Syntax: EINT

Notes: If the microprocessor interrupts have previously been disabled by a DINT command, they may be re-enabled using the EINT command. If EINT is encountered and the interrupts are already enabled, there will be no change.

For more information on EINT, see DINT in Reference Section.

Examples:

```
EINT          'All interrupts are enabled
```

ELOOP

Note: ELOOP is the same as NEXT.

EOF

Type: Numeric Function

Purpose: Returns the value of zero (Ø) if a disk file is open <u>and</u> there is data in the file that has not been read in; otherwise, will return value of one (1).

Syntax: EOF

Notes: This function is read like a variable but cannot be set. The normal use of EOF is to test the value for an indication of an "end of data" when reading in a disk file. If an input file is <u>not</u> open, EOF will equal one (1).

Examples:

```
IF EOF=1 THEN GOTO DONE     'If the end of the current file has been reached,
                            'the program will go to label "'DONE".
IF EOF=0 THEN INPUT A(X)    'If the end of the file has not been reached, the next
                            'value in the file is read into array element A(X).
```

FOR

Type: Statement

Purpose: Allow a blocked group of statements to be executed a given number of times.

Syntax: FOR {var}={beginning var. value} TO {ending var. value}

{var} - Numerical variable. (Array elements are **not** allowed)

{beginning var. value} - Value initially placed in the referenced variable.

{ending var. value} - Value that the referenced variable will increment up to.

Notes: The beginning value and the ending value must be within the range of +/-32766. Both "beginning var. value" and "ending var. value" may be numerical expressions. When this statement is first encountered, the referenced variable is initially set to the integer of the beginning variable value. When the program reaches a NEXT or ELOOP statement, the value of the variable is incremented by one and tested against the ending variable value. If the value is larger than the ending value, the program proceeds to the statement after the NEXT statement. If, after incrementing, the variable is less than or equal to the ending value, the program "jumps" to the program statement following the FOR statement, and proceeds. FOR/NEXT loops may be "nested" within each other up to eight (8) loops deep. FOR and LOOP are exactly the same statement, as are NEXT and ELOOP. and may be freely interchanged.

Examples:

```
FOR VAR1 = 4 TO 8          'Display will show "4 5 6 7 8"
DISP VAR1,
NEXT
```

GET

Type: Program Command

Purpose: Read one character through serial input (RS-232)

Syntax: GET {serial port selector;}{string variable},{string variable}, ...

{serial port selector;} - (Optional) "@" followed by desired serial port# (1-4), followed by semicolon. This may be a numerical expression. (Note: Only needed if 1041 Serial Port Expander is used)

{string variable} - Any valid string variable

Notes: This command is used to read in one character through the serial communications port. The GET command treats incoming characters differently, depending on whether the Direct Command Mode is enabled or disable.

GET (Continued)

Direct Command Mode Enabled

When the program reaches the GET command, the serial input buffer is cleared and the program will wait for an incoming character. The first character is read in and placed in the referenced string variable. There is no limit on the amount of time the FT-100a will "wait" for an incoming character. If no character is received, it can appear the program is "locked up". This situation is also aggravated by the fact that all external interrupts are disabled while the FT-100a is waiting. Since the front panel pushbuttons utilize interrupts, they will appear to be inoperative as well. The only way to exit the GET command with Direct Command Mode enabled is to either input a character through the serial port or press the RESET pushbutton to abort.

Direct Command Mode Disabled

All serial data is placed in the serial input buffer in the order in which it is received. The GET command will extract one character from the buffer, in the same order in which it was input (first in, first out). If there are no characters in the buffer, a "null" character will be returned. (The buffer can hold up to 128 characters before an overflow error is received)

This is normally considered the best mode in which to use the GET command. However, understand that while the Direct Command Mode is disabled, the operator can have no access or control through the serial port. For more information about the Direct Command Mode, see the COM command in the Reference Section.

If the GET command contains more than one string variable, characters will be input and placed in the respective string variables. There is no processing done to the incoming character. Only ASCII characters may be input. After all referenced string variables have been input, the program will move on to the next statement.

If an optional 1041 serial port expander is attached, it may be switched by specifying the serial port# in the GET command (This may also be done in the PRINT or IN commands). The serial port may be selected by a variable or numerical expression. When power is turned on, a RESET command issued, an error encountered, or the program stopped or started, serial port# 1 is always the default value. If the program changes serial ports, it will remain changed for the duration of the program unless changed by another command. For this reason, if a data terminal or computer is to be connected, it should normally be on serial port# 1. For more information, see the serial port expander manual.

Examples:

```
GET A$                  'Program stops & waits for a character to be input through the
                        'serial input, and places input character in string variable A$
GET A$,B$               'First character is placed in A$, next character is placed in B$.
GET VAR1                'ERROR! Only string characters may be input.
GET @3;A$,B$,@2;C$      'Input two characters through port# 3. Put the 1st into
                        'A$ & the 2nd into B$. Then input a character through
                        'serial port# 2 & place in C$.
```

GETKEY

Type: Program Command

Purpose: Read one character from keyboard input

Syntax: GETKEY {string variable},{string variable}, ...

{string variable} - Any valid string variable

Notes: This command is used to read in one character from the keyboard input. The GETKEY command reacts differently, depending on whether the Direct Command Mode is enabled or disable.

Direct Command Mode Enabled

When the program reaches the GETKEY command, the keyboard buffer is immediately cleared and the program, waits for an incoming character. The first keyboard character is read in and placed in the referenced string variable. There is no limit on the amount of time the FT-100a will "wait" for an incoming character. If no character is received, it can appear the program is "locked up". This situation is also aggravated by the fact that all external interrupts are disabled while the FT-100a is waiting. Since the front panel pushbuttons utilize interrupts, they will appear to be inoperative as well. The only way to exit the GETKEY command with Direct Command Mode enabled is to either input a keyboard character or press the RESET pushbutton to abort.

Direct Command Mode Disabled

All keyboard input data is placed in the keyboard buffer in the precise order in which it is received. The GETKEY command will extract one character from the buffer, in the same order in which it was input (first in, first out). If there are no characters in the buffer, a "null" character will be returned. (The keyboard buffer can hold up to 128 characters before an overflow error is received)

This is normally considered the best mode in which to use the GETKEY command. However, while the Direct Command Mode is disabled, the operator has no access or control through the keyboard. For more information about the Direct Command Mode, see the COM command in the Reference Section.

If the GETKEY command contains more than one string variable, characters will be input and placed in the respective string variables. There is no processing done to the incoming character. Only ASCII characters may be input. After all referenced string variables have been input, the program will move on to the next statement.

Examples:

GETKEY A$	'Program stops & waits for a character to be input through the 'serial input, and places input character in string variable A$
GETKEY A$,B$	'First character is placed in A$, next character is placed in B$.
GETKEY VAR1	'ERROR! Only string characters may be input.

GOTO

Type: Program Command

Purpose: Cause program to immediately jump to a given location (Label)

Syntax: GOTO {label or line number}

{label or line number} - Label name specifying destination of next program statement. This may also be a line number. String expressions are not allowed as labels.

Notes: Program will immediately jump to the location of the referenced label or line#. The label may be anywhere in the program.

Caution: If GOTO references a line number, be aware that the line number reference will not be updated if the program is edited! Use of line number references is most useful for direct command operation.

Examples:

```
GOTO LABEL1                    'Immediately jump to LABEL1
IF X = 1 THEN GOTO LABEL2      'If variable X=1, then jump immediately to LABEL2
GOTO 39                        'Immediately jump to line# 39
```

IF . . . THEN

Type: Conditional Statement

Purpose: Test a condition and branch accordingly

Syntax: IF {comparison expression} THEN {command line}

{comparison expression} - Two or more expressions with comparison operators. Multiple comparison expressions may be joined in the same statement by boolean operators.

{command line} - Any valid line of program commands or statements.

Notes: The comparison expression is tested, and if "true", the command lines after THEN are executed as the next lines of the program. If the comparison expression is not true, all commands after THEN are "skipped" and the program "falls through" to the next program statement following the IF...THEN statement.
Comparisons are consider "not true" if, and only if, they evaluate to a value of zero (Ø).

The comparison expression may have multiple levels of parenthesis as well as all valid string and number functions. Multiple comparisons may be made by using boolean operators.

Examples:

```
IF VAR1 = VAR2 THEN BEEP
IF ((3.4 + 6.9)/4) -1 = 1.575 THEN DISP "CORRECT!": END
IF "AB" + "CD" = A$ THEN GOTO LABEL1
IF (A=B AND C=D) OR E=F THEN CALL SUB1
```

IN

Type: Program Command

Purpose: Read character string through serial input (RS-232)

Syntax: IN {serial port selector;}{string variable},{string variable}, ...

{serial port selector;} - (Optional) "@" followed by desired serial port# (1-4), followed by semicolon. This may be a numerical expression.
(Note: Only needed if 1041 Serial Port Expander is used)

{string variable} - Any valid string variable

Notes: This command allows the program to stop and wait for an incoming character string through the serial communications port. The FT-100a will input characters until a carriage return (ØDH) is received. Then the string is placed in the referenced string variable. If more than one string variable is specified, the process will be repeated. Only character strings may be input. Numbers are input as strings and may be converted back to numbers by using the VAL function (see VAL in Reference Section).

If an optional serial port expander is attached, it may be switched by specifying the serial port# in the IN command (This can also be done in the PRINT command). Serial port may be selected by a variable or numerical expression. When power is turned on, a RESET command issued, an error encountered, or the program stopped or started, serial port# 1 is always the default value. If the program changes serial ports, it will remain changed for the duration of the program unless changed by another command. For this reason, if a data terminal or computer is to be connected, it should normally be on serial port# 1. For more information, see the serial port expander manual.

Examples:

```
IN A$,B$              'Program waits for input from the serial input, and places input string in
                      'string variable A$
IN VAR1               'ERROR! Only string variables may be input through the serial input.
IN @3;A$,B$,@2;C$     'Input variables A$, then B$ from serial port# 3, then input
                      'variable C$ from serial port# 2
```

INC

Type: Numerical Command

Purpose: Add one (1) to an entity

Syntax: INC {number variable, numerical array, or digital output port}

Notes: This command is a convenient way to increase the valure of an entity by one. An equate statement can achieve the same results. If a digital output port is referenced, and the value was 255 (0FFH), the new value will be zero (Ø).

Examples:

```
INC VAR1              'Add one to value in variable VAR1
INC PC5               'Increment value at digital output port #5 in module "C"
```

INKEY

Type: Program Command

Purpose: Read character string from keyboard input

Syntax: INKEY {string variable},{string variable}, ...

{string variable} - Any valid string variable

Notes: This command allows the program to stop and wait for an incoming character string from the FT-100a keyboard. Upon reaching this command, the program will immediately halt, and clear the keyboard input buffer. Keyboard characters will be input until a carriage return (ØDH) character is received. The string will then be placed in the referenced string variable. If more than one string variable is specified, the process will be repeated. Only character strings may be input. Numbers are input as strings and may be converted back to numbers by using the VAL function (see VAL in <u>Reference Section</u>).

Examples:

```
INKEY A$           'Program waits for input from the keyboard, and places input
                   'string in string variable A$
INKEY X$,Y$,Z$     'Multiple inputs
```

INPUT

Type: Disk File Command

Purpose: Read data from disk file

Syntax: INPUT {variable},{variable},.....

{variable} - Numerical or string variable (including arrays)

Notes: In order to read data from a disk, the file <u>must</u> have been opened as an input file (using the OPEN command).

Since data was stored on the disk in sequential order, it will be input in the same order. This makes it very important to know the order of the data on the disk file. Since numerical and/or string data may be stored in the same disk file, the data must be input into the proper variable type.

Data is always stored on a disk file as ASCII characters. Numbers are stored as their ASCII equivalent. If data is input into a numerical variable, it will be converted into its numerical equivalent. If the data is a non-number string, the value will be zero (Ø). If number data is input into a string variable, the string variable will contain the string equivalent of the number.

Several data elements may be read in from a disk file with one INPUT command. Each variable must be separated by a comma (,) delimiter.

Examples:	INPUT VAR1, VAR2	'Read in numbers into variables VAR1 and VAR2
	FOR X = 1 to 100 INPUT A(X) NEXT	'Loop to read 100 values into array A(x)
	LAB1: IF EOF=0 THEN INPUT VAR1:GOTO LAB1	'Test for end-of-file condition

INT

Type: Numerical Function

Purpose: Calculate and return the integer portion of a numerical expression.

Syntax: INT ({numerical expression})

{numerical expression} - Any valid numerical expression

Notes: An integer is the truncated value of any number. In other words, the number is rounded down to the nearest whole number.
Negative numbers are also rounded down.

Examples:	X = INT 3.14159	'Set variable X to 3
	VAR1 = 6.7	
	DISP INT (VAR1 - 26)	'Display -20 (Integer of -19.3 is -20)

INV

Type: Numerical Command

Purpose: Invert all binary bits of an entity

Syntax: INV {numerical variable, array element, digital output port or line}

Notes: This command will invert all bits of a variable, array element, digital output port, or digital output line. Because of the way negative numbers are stored and represented in a computer, the decimal representation of a number that has been inverted can be confusing. The decimal representation of an inverted number will be the original number with its sign changed and one (1) subtracted from the result (2's compliment). For example, if we invert the number 5, the result is -6.

Examples:	VAR2 = 1101B	'Set VAR2 to 13
	INV VAR2	'VAR2 will = 1111111111110010B or -14
	INV PA3	'Invert all lines in digital port #3 in module "A"
	INV LA(26)	'Invert digital output line# 26

INX

Type: Program Command

Purpose: Read character string from external module

Syntax: INX {module reference,}{string variable},{string variable}, ...

{module reference} - (Optional) "@" followed by desired module reference (A-H).

{string variable} - Any valid string variable

Notes: This command will read a character string from the selected external module, into the designated string variable. If more than one string variable is specified, it will be input as well.

This command requires an external module that supports INX/OUTX communications. The manual with each external module will contain information on the use of INX and OUTX, if supported.

Once the module reference has been set, it will remain in effect until changed by another INX or OUTX command. The <u>only</u> way to change the module reference is through the INX or OUTX commands. There is no default.

Examples:

INX @D,A$,B$	'Input character strings from external module "D" and place in 'string variables A$ and B$.
INX C$	'Input character string using the last selected module reference

LA, LB, LC, LD, LE, LF, LG, LH

Type: Digital Input/Output Line Reference Designators

Purpose: Directly address digital input and output lines as variables.

Syntax: L {module}{line#} or L {module}{port#},{port line#}

{module} - Module designator A-H. Each module has its own unique designator.

{line#} - Digital line number within the referenced module.
Every digital input/output line, within a module, is numbered, beginning with 1 to the total number of digital input and/or output lines within the module. Digital port #1 has lines #1-8, port #2 has lines #9-16, and so on.

{port#} - The digital port number, within a given module.

{port line#} - The line number (1-8) within the referenced digital input or output port.

Notes: Line reference designators allow addressing individual digital input and/or output lines directly, like a variable. The value read from a line reference designator will always equal zero (Ø) or one (1).

If a digital line is set to any value other than zero (Ø), the digital line will equal the least significant bit of the value.

All digital input and output lines within a module have line numbers. Within each module, the digital output ports are always referenced first, then the input. In other words, the output ports are numbered beginning with 1 and ending with the total number of output ports. The digital input ports are then numbered beginning with the next larger number. For example, if module "D" has three digital input and three digital output ports, the output ports are referenced as PD1, PD2, PD3, and the input ports are referenced as PD4, PD5, PD6. (Note, normally there are the same number of digital input and output ports)
The digital lines are numbered in the same manner as the digital ports. Using the above example, the output lines are referenced as LD1-LD24, and the input lines are referenced as LD25-LD48.

If the digital lines are referenced as {port#},{port line#}, the line number is 1-8 for each individual port. Once again, using the above example, the output lines will be referenced as LD1,1-LD1,8, LD2,1-LD2,8, LD3,1-LD3,8 and the input lines are referenced as LD4,1-LD4,8, LD5,1-LD5,8, LD6,1, LD6,8.

Note that in this example the following reference designators are the same:

```
LD1   =   LD1,1
LD2   =   LD1,2
      .
      .
LD8   =   LD1,8
LD9   =   LD2,1
      .
      .
```

For more information, look in the Programming Section under Addressing I/O.
For information on digital ports, look at PA, PB, PC, PD, PE, PF, PG, PH in the Reference Section.

In either type of digital line reference, references may be done using variables or numerical expressions. In this case, the reference must be in parenthesis.

Examples:

```
LA3=1             'Set digital output line #3 in the main module to  "1" (high).
VAR1=LF5+PA2      'Read digital input line #5 in module "F" and add the value to
                  'digital output port 2 in main module, then place the result in VAR1
LB4,6 = 0         'Set digital output line #6 in port #4 in module '"B" to zero (Ø).
                  '(Note:  This is the same line as LB30)
LB(VAR1)=1        'Line reference by a variable
DISP LD(P,L+4)    'Port#,Line# references by numerical expressions
```

LEFT$

Type: String Function

Purpose: Generate and return a character string comprised of the leftmost characters within the referenced string, and being a given number of characters in length.

Syntax: LEFT$ ({string expression},{number of characters})

{string expression} - any valid string expression

{number of characters} - The number or characters of the string expression to be returned. This may be any numerical expression. Note, if the number of characters to be returned is more than the number of characters in the referenced string, the entire string will be returned.

Examples:

```
DISP LEFT$ ("ABCDEFG",3)          '"ABC" is displayed
VAR1 = VAL (LEFT$ (DATE$,2))      'extract string from real-time clock/calendar,
                                  'comprising the month, convert to number, and
                                  'place in variable VAR1
```

LEN

Type: String Function

Purpose: Returns the number of characters in an ASCII character string

Syntax: LEN {string expression}

Notes: "Spaces" are considered characters in strings.

Examples:

```
DISP LEN "House"            'Display 5
A$ = "Y-tek, Inc. FT-100a"  '(18 characters)
VAR1 = 36/LEN A$            'Set variable VAR1 to 2. (36/18)
```

LOOP

NOTE: LOOP is the same as FOR. See FOR statement.

LRA, LRB, LRC, LRD, LRE, LRF, LRG, LRH

Type: Program Command

Purpose: Reset the latches on Externally Triggered Input Ports

Syntax: LR{module}{port number}

{module} - FT-100a module (A-H). The FT-100a main unit is always module A.

{port number} - The individual port number for the externally triggered digital input port.

Notes: This command is used to reset externally gated ports. This is a one-word command that does not need any parameters. The polarity of externally triggered latches is set with the TP{module} commands. Latch Reset is independent of the Trigger Polarity.

If the referenced port number is a numerical expression, it must be within parenthesis.

It is normally recommended that all gated I/O ports be reset after trigger polarities are set and just before the gated port is to be used. This insures a known condition at the latched port.

For more information, look in the Programming Section under Addressing I/O.

Examples:

```
LRA1            'Reset external latched port in the main module.
LRC3            'Reset external latched port #3 in module "C".
LRF(VAR1+VAR2)  'Port reference by a numerical expression.
```

LSA, LSB, LSC, LSD, LSE, LSF, LSG, LSH

Type: Status Indicator

Purpose: Read the Latch Status of any gated Digital Input Ports.

Syntax: LS{module}{port number}

{module} - FT-100a module (A-H). The FT-100a main unit is always module A.

{port number} - The individual port number for the gated digital input port.

Notes: Latch status information is read like any other variable but cannot be set. If the referenced digital port has been triggered, the latch status will equal one (1), otherwise it will equal zero (Ø). Any attempt to set the latch status will result in an "Improper I/O Reference" error.

It is normally recommended that all gated I/O ports be reset after all gate polarities are set and just before the gated port is to be used. This insures a known condition at the externally triggered port.

If the referenced port number is a numerical expression, it must be within parenthesis.

For more information, look in the Programming Section under Addressing I/O.

Examples:	DISP LSA1	'Display latch status for the gated digital port in the main module.
	VAR1=VAR2+LSC4	'Read the latch of gated port #4 in module "C" and add to the 'variable VAR2 and 'put result in 'variable VAR1.
	LSE3=1	'Error! Latch status information can be read only!
	LSG(VAR1)	'Read the latch of gated port referenced by VAR1 in module "G"

MID$

Type: String Function

Purpose: Generate and return a character string beginning with the first referenced character in a character string and being a given length.

Syntax: MID$ ({string},{1'st char number},{number of chars})

{string} - Any valid string expression.

{1'st char number} - The number of characters from the left of the referenced character string to the 1st character of the desired string. This may be a numerical expression.

{number of characters} - The number or characters of the string expression to be returned. This may be a numerical expression.

Notes: If the number of characters to be returned is more than the remaining characters in the referenced string, the remaining characters will be returned. If the first character to be returned is more than the number of characters in the string, a null string will be returned.

Examples:	DISP MID$ ("ABCDEFG",3,2)	'Display "CD"
	VAR1 = VAL (MID$ (DATE$,2,2))	'Read real-time clock/calendar, extract string 'comprising the day, convert to number, and place in variable VAR1.

NAND

Type: Logical operator

Purpose: Used in Boolean operations to compare bits. Will cause the logical NAND of two numerical expressions.

Syntax: {numerical expression} NAND {numerical expression}

Notes: Logical Operators may operate only on Integers. All floating point numbers will be converted into integers before the logical operation. This truncates all fractional parts of the numbers. Since Integers are always 16 bits in length, any Logical operation will consider all 16 bits, including leading zeros (Ø) and the sign bit. This is very important to consider any time Logical Operators are used.

Logical Operators operate on the "bits" of a number. Because of the way negative numbers are represented in digital computers, the results of a logical operation involving a negative number can appear confusing if the result is shown in "decimal" format (base 10). Viewing the numbers in binary format (base 2) shows each bit of the numbers and the result.

For more on Logical Operators, see Operator Types in the Programming Section.

Examples:

```
VAR1 = 1011B              ' 1011B = 11
VAR2 = 1101B              ' 1101B = 13
X = VAR1 NAND VAR2        'Result is 1011B NAND 1101B = 1111111111110110B
DISP VAR1 NAND 6          'Display -3 (decimal format) or 1111111111111101B
```

Note: NAND can be used in IF/THEN statements to combine comparisons:

```
IF ((PA6 NAND VAR1) = VAR2) OR PA3=5CH THEN GOTO LABEL1
```

NEG

Type: Numerical Function

Purpose: Calculate and return the negative value of a numerical expression.

Syntax: NEG ({numerical expression})

{numerical expression} - Any valid numerical expression

Notes: This function merely inverts the sign of a numerical expression. A positive numerical expression returns a negative number, and a negative numerical expression returns a positive number.

Examples:

```
X = NEG 45.67             'Set variable X to -45.67
VAR1 = -8.32
DISP NEG (VAR1 + 43)      'Display -34.68
```

NEXT

Type: Statement

Purpose: Mark end of a FOR/NEXT loop. (See FOR statement)

Syntax: NEXT {loop variable}

{loop variable} - (optional) The variable that is used to count the number of times the program has gone through the FOR/NEXT loop.

Notes: When the program reaches a NEXT statement, the referenced loop variable is incremented by one and tested against the ending value. If the referenced loop variable is larger than the ending value, the program will "fall through" to the next statement/command following the NEXT statement; otherwise, the program will "loop" back to the statement following the FOR statement.

The "loop variable" reference may be included after the NEXT statement solely to make the program more understandable, but it is ignored by the FT-100a. If the "loop variable" reference is incorrect, no error will result.

NEXT is exactly the same statement as ELOOP and may be freely interchanged.

For more information and examples, see FOR in Reference Section.

NO

Type: Operator Input

Purpose: Provide direct operator feedback

Syntax: NO

Notes: NO is actually a numeric function that can have a value of one (1) or zero (Ø), depending on whether or not the NO pushbutton is pressed. NO will equal zero (Ø) if the pushbutton is not pressed, and will equal one (1) if the pushbutton is pressed.

The FT-100a reads the NO pushbutton at the precise moment that the NO function is evaluated. There is no wait or delay. The front panel ESC pushbutton is electrically tied to the NO pushbutton and they may be used interchangeably.

This function, along with the YES function, provides a very simple method of operator feedback to simple questions. YES and NO may be used in IF...THEN statements or they may be used in numerical expressions just like any other function. Note, unlike most other functions, YES and NO do not have any parameters.

NO (Continued)

Examples:	IF NO THEN GOTO BEG	'If NO pushbutton is pressed, jump to label BEG
	VAR1=NO*3.4	'NO can be used like any function

This routine asks the operator a question and waits for the result:

```
    DISP "  Is the ": DISP " RED LED on?": DISP "Press YES or NO"
L1: IF YES THEN BEEP: GOTO L2
    IF NO THEN CALL ERR1: GOTO L3
    GOTO L1
```

The operator is instructed to observe the red LED. If it is on (YES pressed), the FT-100a will beep and go on to label L2. If the LED is not on (NO pressed), the error routine ERR1 is called and routine goes to label L3. The routine sits in a loop, waiting for the YES or NO response.

NOR

Type: Logical operator

Purpose: Used in Boolean operations to compare bits. Will cause the logical NOR of two numerical expressions.

Syntax: {numerical expression} NOR {numerical expression}

Notes: Logical Operators may operate only on Integers. All floating point numbers will be converted into integers before the logical operation. This truncates all fractional parts of the numbers.

Since Integers are always 16 bits in length, any Logical operation will consider all 16 bits, including leading zeros (Ø) and the sign bit. This is very important to consider any time Logical Operators are used. Logical Operators operate on the "bits" of a number.

Because of the way negative numbers are represented in digital computers, the results of a logical operation involving a negative number can appear confusing if the result is shown in "decimal" format (base 10). Viewing the numbers in binary format (base 2) shows each bit of the numbers and the result.

For more information about Logical Operators, see Operator Types in the Programming Section.

Examples:	VAR1 = 1011B	' 1011B = 11
	VAR2 = 10B	' 10B = 2
	X = VAR1 NOR VAR2	'Result is 1011B NOR 10B = 1111111111110100B
	DISP 4 NOR 2	'Display -7 (decimal format) or 1111111111111001B (See Notes for explanation of results)

Note: NOR can be used in IF/THEN statements to combine comparisons

IF ((PA6 NOR VAR1) = VAR2) OR PA3=5CH THEN GOTO LABEL1

NOT

Type: Numerical Function

Purpose: Invert all binary bits of a number. (Creating the one's compliment of the number)

Syntax: NOT ({numerical expression})

Notes: If the numerical expression is an integer, all bits in the number are inverted, yielding the one's compliment of the number. If, however, the expression is a floating point number, the function will invert the sign of the number and subtract one from the result. This is because of the way negative numbers are stored in computers.

The NOT function does not change any of the entities of the numerical expression.

Examples:

```
X = NOT 26                'Set variable X to -27
DISP NOT 3.45             'Display -4.45
DISP #B8; NOT 101101B     'Display 11010010B
```

ON . . . GOTO

Type: Conditional Statement

Purpose: Compare the value of a variable and branch accordingly

Syntax: ON {numerical variable} THEN {label1, {label2, {label3, ...

{numerical variable} - Any valid numerical variable or numerical expression.

{label1 - Destination label if integer of numerical variable evaluates to one (1).
{label2 -Destination label if integer of numerical variable evaluates to one (2).

-
-

Notes: The numerical expression is evaluated, and if the integer of the result is equal to one (1), the program will immediately jump to the command line at label1. If the expression evaluates to two (2), the program will jump to label2. This would continue on for 3, 4, 5, etc. for as many labels as the ON command indicates. If the numerical expression evaluates to zero, a negative number, or an integer larger than the number of labels, the program will "fall through" to the next statement immediately following the ON command.

Examples:

```
ON VAR1 GOTO LAB1,LAB2,DONE     'Jump to LAB1 if VAR1=1, jump to LAB2 if
                                'VAR1=2, jump to DONE if VAR1=3
```

OPEN

Type: Disk File Command

Purpose: Permit access to a disk file.

Syntax: OPEN {filename},{filetype}{,APPEND (optional)}

{filename} - Name of file to be accessed. Filenames must be a character string beginning with A-Z and can be up to 8 characters plus decimal point and 3 extension characters (decimal point and extension are not required). Filename must be in quotes unless it is a string expression.

{filetype} - Type of file accessed ("IN" for input and "OUT" for output)*

{,APPEND} - (Optional) If the file is an existing output file, placing "APPEND" in the OPEN command will cause all new data to be "appended" to the data that is currently in the referenced file.*

* IN, OUT, and APPEND may be abbreviated using their first letter.

Notes: In order to access a disk file, for input or output, the file must be "opened". Likewise, when access to a file is complete, the file must be "closed" using the CLOSE command. Files left open can be inadvertently changed or destroyed. It is always considered good programming practice to "close" all unused files.

Only one file can be open at any given time.

If an input file is open, each time the INPUT command is encountered, the next data element is taken from the file.

If an output file is open, each time the STORE command is executed, the data is sequentially stored in the referenced file. If the "append" option is enacted, the data will be placed at the end of the existing data. If an existing output file is opened, without the "append" designation, all previous data in the file will be destroyed.

Examples:

```
OPEN "FILE1",IN       'Open input file "FILE1"
OPEN "FILE2",OUT      'Open file "FILE2" as an output file.
                      '(If this file already exists,'the old data will be overwritten)
OPEN "FILE3",OUT,A    'Open output file "FILE3" and append all new data to end
                      'of existing data.
```

OR

Type: Logical operator

Purpose: Used in Boolean operations to compare bits. Will cause the logical OR of two numerical expressions.

Syntax: {numerical expression} OR {numerical expression}

Notes: Logical Operators may operate only on Integers. All floating point numbers will be converted into integers before the logical operation. This truncates all fractional parts of the numbers. Since Integers are always 16 bits in length, any Logical operation will consider all 16 bits, including leading zeros (Ø) and the sign bit. This is very important to consider any time Logical Operators are used.

Logical Operators operate on the "bits" of a number. Because of the way negative numbers are represented in digital computers, the results of a logical operation involving a negative number can appear confusing if the result is shown in "decimal" format (base 10). Viewing the numbers in binary format (base 2) shows each bit of the numbers and the result.

For more on Logical Operators, see Operator Types in the Programming Section.

Examples:

```
VAR1 = 1100B                  ' 1100B = 12
VAR2 = 1001B                  ' 1001B = 9
X = VAR1 OR VAR2              'Result is 1100B OR 1001B = 1101B
DISP (VAR2 OR 2) AND 14       'Display 10  [(9 OR 2) AND 14]
```

Note: OR can be used in IF/THEN statements to combine comparisons

```
IF ((A=(B+3)) AND A$="ABC") OR PA3=5CH THEN GOTO LABEL1
```

OUTX

Type: Program Command

Purpose: Output character string to external module

Syntax: OUTX {module reference,}{expression},{expression}, ...

{module reference} - (Optional) "@" followed by desired module reference (A-H).

{expression} - String or numerical expression (numerical expression must evaluate to a value from Ø to 255, inclusive)

Notes: This command will send either a character string or an 8-bit number to the selected external module. If more than one expression is specified, it will be output as well.

If a numerical expression is to be output, it must be between Ø-255 (inclusive). Strings are output one character at a time, in serial order, left to right.

This command requires an external module that supports INX/OUTX communications. The manual with each external module will contain information on the use of INX and OUTX, if supported.

Once the module reference has been set, it will remain in effect until reset by another INX or OUTX command. The <u>only</u> way to change the module reference is through the INX or OUTX commands. There is no default.

Examples:	OUTX @D,A$,B$	'Send character strings to external module "D"
	OUTX C$,VAR1,3CH	'Send character string & two numbers using the last selected 'module reference

PA, PB, PC, PD, PE, PF, PG, PH

Type: Digital Input/Output Port Reference Designators

Purpose: Directly address digital input and output ports as variables.

Syntax: P{module}{port number}

{module} - FT-100a module (A-H). The FT-100a main unit is always module A.

{port number} - The individual port number for the digital input or output port.

Notes: Digital input and output ports are always eight (8) bits and are addressed like any other variable. The second letter in the designator indicates the module where the digital port is located. Module designators are set by "DIP" switches or jumpers, and are unique for each separate module. The third digit (number) is the individual port number. These numbers start at 1 and progress upward to the maximum number of digital ports in the module. The reference port number may be a variable or numerical expression in which case it <u>must</u> be in parenthesis.

Output ports may be set to any value from 0 to 255. Input ports are "read only" and cannot be set to a value.

For more information, see <u>Addressing I/O</u> in the <u>Programming Section</u> or <u>Hardware Interfacing</u> in the Appendix.

Examples:	PA3=45H	'Set digital output port 3 in the main module to 45H (hex)
	VAR1=PD5+PA2	'Read digital input port 4 in module "D" and add 'to value of digital 'digital output port 2 in main 'module, then place result in VAR1
	PA7=128	'Error! PA7 is an ~~output~~ port and cannot be set. ~~Output~~ ports 'can only be read.
	DISP PG(VAR1+VAR2)	'Port# reference by numerical expression.

PEEK

Type: Numeric Function

Purpose: Retrieve the numerical representation of the contents of a given memory location.

Syntax: PEEK {memory address}

{memory address} - Numerical value indicating the memory address to be read. The memory address must be between Ø and 131,071 (inclusive) and may be a numerical expression.

Notes: This function retrieves the contents of a memory location within the FT-100a. The value will always be between Ø and 255_{10} (Ø-ØFFH) since the FT-100a uses 8-bit wide memory.

Examples:

```
DISP PEEK 5943                   'Display the contents of memory location 5943
VAR1 = PEEK 94055 + PEEK 7334    'Set variable VAR1 to the sum of the contents
                                 'of memory locations 94055 and 7334.
```

POKE

Type: Program Command

Purpose: Place a given value in one byte of memory within the FT-100a

Syntax: POKE {memory address},{value}

{memory address} - Numerical value indicating the memory address for the destination of the value to be inserted. The memory address must be between Ø and 131,071 (inclusive) and may be a numerical expression.

{value} - The number to place in the referenced memory location. Value must be between Ø and 255 (inclusive) and may be a numerical expression.

Notes: This command, along with the PEEK command, allows the programmer access to the internal memory of the FT-100a. Care should be taken any time internal memory values are changed. Be sure of the results before executing a POKE command!

Examples:

```
POKE 4300H,0D4H     '0D4H (212₁₀) is placed at memory location 4300H (17152₁₀)
POKE VAR1,VAR2      'Place VAR2 in memory location determined by VAR1
```

PRINT

Type: Program Command

Purpose: Send numerical or string data out serial communications port (RS-232)

Syntax: PRINT {format;}{serial port selector;}{expression}{del}{expression}{del} .....

{format;} - (Optional) Changes print format. The number format procedure is exactly the same as for the DISP command. See DISP in Reference Section.

{serial port selector;} - (optional) Change active serial port. "@" followed by desired serial port# (1-4). If a serial port expander is attached to the FT-100a, the active serial port may be selected with this command. A semicolon must follow this option. The active serial port may be selected at any location within PRINT command. This may be a numerical expression.

{expression} - Any valid numerical or string expression

{del} - (delimiter) Data separator. Type of delimiter determines spacing between expressions when printed. Delimiter Types:

Semicolon (;) - Leave no space between expressions.

Comma (,) - Separates display into columns, each 15 characters wide. (A comma will effectively "tab" over to the next column.)

Notes: The PRINT command is very similar to the DISP command, but with the result going out the serial communications line.

{format} and {serial port selector} are optional and either may occur anywhere within the PRINT command. This allows changing number formats and/or serial ports several times within the same command. A numerical expression may be used for selecting the serial port, but not for format.

"PRINT" may be abbreviated by using the question mark ("?").

In order to have multiple serial ports, a serial port expander must be installed. When power is turned on, a RESET command issued, an error encountered, or the program stopped or started, serial port# 1 is always the default value. If the program changes serial ports, it will remain changed for the duration of the program unless changed by another command. For this reason, if a data terminal or computer is to be connected, it should normally be on serial port# 1. For more information, see the serial port expander manual.

Examples:

```
PRINT VAR1,"ABC"          'Print variable VAR1 and string "ABC" spaced out to
                          'next column location
PRINT @1;VAR1;@2;VAR1     'Print VAR1 out serial line#1, and line#2
PRINT VAR1;#B;VAR1        'Print VAR1 in decimal notation and again in binary
? VAR1*VAR2               'Print mathematical product of VAR1 and VAR2
```

READ

Type: Program Command

Purpose: Permit the entry of multiple numerical and/or string values in sequence.

Syntax: READ {variable},{variable}, ...

{variable} - Numerical or string variable or numerical array element.

Notes: The READ command sequentially inputs data supplied by DATA statements. One data element is read in sequentially each time a READ command is executed. Once a data element has been read in, it cannot be read in again unless a CLR command is executed.

DATA statements may be located anywhere within the program but must be in the order that the data will be used.

Numerical and string data may be intermixed within the same DATA statement. If this is done, it is important that the variables being "read into" are of the correct type (numerical or string).

For more information, see DATA in Reference Section.

Examples:

```
READ VAR1, VAR2, VAR3        'Read the next 3 values into VAR1, VAR2, and VAR3
DATA 1.5,3904.56,-3.4e-12    'Data Statement

DATA 2,4,6,8                 'Data statement can be anywhere
FOR J = 1 to 3               'Loop to read in 4 values into array ARRAY (x)
READ ARRAY(J)
NEXT

READ A$,B$,VAR1                  'Read 2 strings into string variables A$ & B$, and
                                 'number into VAR1
DATA "DOG","HOUSE",3.14159          'Data statement
```

RETURN or RET

Type: Program Command

Purpose: Marks the end of a subroutine. Instructs program to jump to the statement following the CALL command.

Syntax: RETURN or RET

Notes: Variables, arrays, etc. are not changed when going into or out of subroutines.

For more information, see CALL in Reference Section.

Examples:

```
SUB1: CLS                    'This subroutine will clear the LCD
      DISP VAR1              'screen and display variable VAR1
      RETURN
```

RIGHT$

Type: String Function

Purpose: Generate and return a character string comprised of the rightmost characters within the referenced string, and being a given number of characters in length.

Syntax: RIGHT$ ({string expression},{number of characters})

{number of characters} - The number or characters of the string expression to be returned. This may be a numerical expression.

Notes: If the number of characters to be returned is more than the number of characters in the referenced string, the entire string will be returned.

Examples:

```
DISP RIGHT$ ("ABCDEFG",3)        'Display "EFG"
VAR1 = VAL (RIGHT$(DATE$,2))     'Read real-time clock/calendar, extract string
                                 'comprised of the year, convert to number, and
                                 'place in variable VAR1.
```

RND

Type: Numeric Function

Purpose: Generate and return a pseudo-random number between the value of Ø (inclusive) and 1 (non-inclusive).

Syntax: RND {(seed number (optional))}

{seed number} - (optional) Number to place into the random number generator formula. Seed number cannot be a numerical expression.

The seed number type will yield the following results:

(omitted) - The next random number uses the last random number as its seed.
Zero (Ø) - The random number will equal the last random number.
negative - The seed number is generated from the real-time clock.
positive - The seed number is generated using the positive seed number selected.

Notes: Although this function actually generates a sequence of numbers based on the seed value, the result is apparently random.

If a negative seed value is used, it must be in parenthesis. The parenthesis are optional for all others.

When the seed value is set, the sequence of generated numbers is constant for that seed value. This ability can be used to randomly test conditions, yet by recording the seed value, be able to duplicate the sequence leading to a failure or condition.

Once the seed value has been set, the pseudo-random sequence is read by referencing RND (with no parameter), just like a variable. This will give the next number in the sequence.

To achieve a specified range of random numbers, use the following formula:
N = RND * (U-L)+L (L=Lower limit, U=Upper limit of range)

Examples:	DISP RND	'Display random number
	VAR1 = RND Ø	'Set variable VAR1 to the same value as the last random 'number generated.
	DISP RND (-3)	'Display random number using the FT-100a real-time clock as 'the seed number.
	VAR1 = RND (358)	'Set seed value to 358

SHL

Type: Bit-manipulation Command

Purpose: Shift the binary bits of a number one bit to the left

Syntax: SHL {number variable, numerical array, or digital output port}

Notes: The SHL command provides an easy way to shift bits in a variable or digital output port. If the operand is a variable, it is effectively multiplied by two (2).

Bits shifted <u>out</u> of the left of a digital port are lost. A zero (Ø) is always shifted <u>into</u> the right bit location.

Examples:	VAR1 = 23	'Set variable VAR1 to 23_{10} (00010111B)
	SHL VAR1	'VAR1 set to 46_{10} (00101110B)
	PA1 = 0D5H	'Set output port A1 to 0D5H (11010101B)
	SHL PA1	'PA1 now equals 0AAH (10101010B)

SHR

Type: Bit-manipulation Command

Purpose: Shift the binary bits of a number one bit to the right

Syntax: SHR {number variable, numerical array, or digital output port}

Notes: The SHR command provides an easy way to shift bits in a variable or digital output port. If the operand is a variable, it is effectively divided by two (2).
Bits shifted <u>out</u> of the right of the entity are lost. A zero (Ø) is always shifted <u>into</u> the left bit location.

Examples:	VAR1 = 23	'Set variable VAR1 to 23_{10} (00010111B)
	SHR VAR1	'VAR1 set to 11_{10} (00001011B)
	PA1 = 0D5H	'Set output port A1 to 0D5H (11010101B)
	SHR PA1	'PA1 now equals 6AH

SIN

Type: Numeric Function

Purpose: Calculate and return the trigonometric sine of a number.

Syntax: SIN ({numerical expression})

{numerical expression} must be in radians.

Notes: If the parameter is a single entity, parenthesis are not required.
To convert degrees to radians, use: RADIANS = DEGREES / 57.295778

Examples:

```
DISP SIN (1.5)              'Display 0.997495
VAR1 = AA9 * SIN (VAR2)     'Find the sine of the variable VAR2, multiply the
                            'result with the voltage at analog input AA9, and place
                            'result in variable VAR1.
```

SPC

Type: String Function

Purpose: Return a character string containing a given number of blank spaces.

Syntax: SPC ({spaces})

{spaces} - Numerical expression equal to the number of spaces in the returned character string.

Notes: If the parameter is a single entity, parenthesis are not required.

Examples:

```
DISP "ABC";SPC 4;"DEF"      'Display "ABC    DEF"
PRINT SPC 5;3.14            'Print "     3.14"
A$="abc"+SPC (5)+"def"      'A$ will = "abc     def"
```

SQR

Type: Numeric Function

Purpose: Calculate and return the square root of a number.

Syntax: SQR ({numerical expression})

Notes: If the parameter is a single entity, parenthesis are not required.

Examples:

```
DISP SQR 81                 'Display 9 (9 = √81)
VAR1 = 63 - SQR AC4         'Find the square root of the voltage at analog input #4
                            'in module "C", subtract that result from 63, and put the
                            'result into variable VAR1.
```

SS (SINGLE-STEP)

Type: System Command

Purpose: Place the FT-100a in the "Single-Step" mode of operation.

Syntax: SS {ON or OFF}

Notes: This command is useful when debugging a program. If single-step is "on" when the program starts, the FT-100a will execute one statement or command and then pause, indicating the next line#. When the CONT pushbutton is pressed (or CONT command issued as a direct command), the next statement will be executed. This will continue until the end of the program or until single-step is turned "off".

Single-step may be turned on and off within a program, or by direct command. Once turned "on", single-step will stay "on" until turned off by SS OFF command or a RESET command. It is sometimes useful to place an SS ON command at the beginning of a suspect area of the program, and turn off single-step by direct command. If "paused" during single-step, and SS OFF is issued as a direct command, pressing CONT will then cause the program to continue with single-step turned off.

Examples:

```
SS ON          'Single-step is enabled
SS OFF         'Single-step is disabled
```

STORE

Type: Disk File Command

Purpose: Store data on disk file

Syntax: STORE {expression},{expression},·····

{expression} - Either numerical expression or string expression

Notes: In order to store data on a disk, the file must have been opened as an output file (using the OPEN command).

All data is stored on a disk file as ASCII strings. Numbers are converted to ASCII strings prior to storing. Each Data item is stored on the disk in sequential order, followed by Carriage Return and Line Feed characters (0DH & 0AH).

Numerical and/or string data may be stored, and even intermixed, however when read back from the disk, the data should be handled in the proper manner.

More than one expression may be stored with one STORE command. Each expression must be separated by a comma (,) delimiter.

Examples:

```
STORE VAR1, VAR2             'Store values in VAR1 and VAR2 on disk
STORE "ABC",A(3,4),65.4      'Store string "ABC", array element A(3,4), and 65.4
STORE VAR1+(3*X),VAR2        'Evaluate an expression & store result on disk file
```

STR$

Type: String Function

Purpose: Generate and return an ASCII character string equivalent to the numerical value.

Syntax: STR$ ({numerical expression})

Notes: If the converted number is negative, the resulting value will have a minus sign (-) preceding the negative value. However, if the number is positive, a space will precede the value. The number to be converted may be in any acceptable number format - decimal, binary, hexadecimal, or exponential. However, the resulting string is <u>always</u> the decimal representation.

Parenthesis are required only if the parameter has more than one entity.

Examples:

```
DISP STR$ (3.14159)          'Display the string, " 3.14159"
A$ = "Abc" + STR$ (-12.3)    'A$ equals "Abc-12.3"
B$ = STR$ 5CH                'B$ equals " 92" (result is always in decimal format)
```

SUPPLY

Type: Program Command

Purpose: Set the output voltage for the programmable power supply.

Syntax: SUPPLY {voltage}

or: SUPPLY = {voltage}

{voltage} - numerical expression which evaluates to a number from Ø to 10, inclusive.

Notes: This is a special command which sets the output voltage of the programmable supply. The voltage may be set to any value from Ø to 10 volts. The voltage will be set to the nearest value in increments of ≈39.2 mv.

The programmable supply voltage value may be read by treating SUPPLY as a numerical variable. Alternatively, the analog output line AA8 is equivalent to SUPPLY.

Examples:

```
SUPPLY 5.0           'Set voltage to 5 volts
SUPPLY AD5 + VAR1    'Set programmable supply to sum of voltage at
                     'analog input AD5 and variable VAR1
SUPPLY = 3.4         'Set voltage to 3.4 volts
DISP SUPPLY          'Display programmable power supply voltage
```

SYS

Type: System Command

Purpose: Direct the program to jump to an assembly language routine.

Syntax: SYS {segment},{offset}

{segment} - (optional) Internal memory segment address used by microprocessor. If omitted, 0000 will be assumed.

(offset} - Offset address within the specified memory segment. (Ø - 65535)

Notes: Offset and segment contain the address of the assembly routine within the FT-100a.

This command allows loaded an assembly level program or subroutine into the FT-100a memory and be accessed from the high-level program. If the assembly routine has a RET statement, the FT-100a will return to the statement after the SYS command and continue the program. This allows intermixing of high-level programming with assembly level programming.

Assembly level routines normally execute much faster and with greater timing accuracy than high-level programs. For very critical timing operations, assembly routines could be considered.

Examples:

```
SYS 2000H,34A9H     'Immediately jump to seg 2000H at offset 34A9H and execute
                    'the assembly routine.
SYS 19393           'Jump to seg 0000 at offset 19393 (4BC1H) and execute the
                    'assembly routine.
```

TAB

Type: String Function

Purpose: Move cursor or printhead to a designated column

Syntax: TAB ({column number})

{column number} - The column number of the destination of the cursor or printhead. This may be a numerical expression.

Notes: If the cursor is already past the selected column, the cursor will move. This is true for the LCD display as well as the serial output (PRINT command). Columns start numbering from left to right, starting with one (1).

This command is used within the DISP and PRINT commands and may be combined with other print/display commands. If other print/display functions or expressions follow or precede the TAB function, they must be separated by a semicolon (;). Parenthesis are required only if the parameter has more than one entity.

Examples:

```
DISP TAB 12;3.49              'Move LCD cursor to column number 12 and display 3.49
PRINT "Result: ";TAB 10;98.6        'Print "Result:   98.6"
```

TAN

Type: Numeric Function

Purpose: Calculate and return the trigonometric tangent of a number.

Syntax: TAN ({numerical expression})

{numerical expression} must be in radians.

Notes: As with all functions, single entity parameters do not require parenthesis.
To convert degrees to radians, use: RADIANS = DEGREES / 57.295778

Examples:

DISP TAN (0.8)	'Display 1.029639
VAR1 = VAR2 + TAN (AA10)	'Add the tangent of the analog input voltage on AA10 'to the number in VAR2. Place result in variable VAR1.

TIME$

Type: Dedicated String Variable

Purpose: Set or read the time-of-day in the realtime clock/calendar

Syntax: TIME$

Notes: TIME$ is a dedicated string variable that is used to set or read the real-time clock in the FT-100a. The format for the variable is very important and must always be followed. The real-time clock may be used as a 12-hour or 24-hour clock. The format of the string used to set TIME$ determines the mode of operation.

When read, TIME$ returns a character string that may be treated like any other string. Note: If there is a need to use any part of the time-of-day in a numerical manner, simply use string functions LEFT$, RIGHT$, or MID$ to extract the desired portion of the string and then use the VAL function to convert the sub-string to a number.

TIME$ string format to use when setting the time
24-hour mode - "hhmm"
12-hour mode - "hh:mm xx" (Note the space between minutes and AM/PM)

TIME$ string format to use when reading the time
24-hour mode - "hhmm"
12-hour mode - "hh:mm:ss xx" (Note the space between minutes and AM/PM)

Definitions:
hh - hours (1-12 for 12-hour, Ø-23 for 24-hour)
mm - minutes (Ø-59)
ss - seconds (Ø-59)
xx - AM or PM (for 12-hour mode only)

Examples:

TIME$ = "12:34 PM"	'Set the real-time clock to 12:34 PM (12 hour mode)
TIME$ = "1234"	'Set the real-time clock to 12:34 PM (24 hour mode)
DISP TIME$	'Display current time in preselected form (12-hour or 24-hour)
DISP LEFT$ (TIME$,2)	'Display the hours only.

TPA, TPB, TPC, TPD, TPE, TPF, TPG, TPH

Type: I/O Setup Command

Purpose: Set the trigger polarity of a externally latched digital input port.

Syntax: TP{module}{port number}

{module} - FT-100a module (A-H). The FT-100a main unit is always module A.

{port number} - The individual port number for the digital input port. This may be a numerical expression in parenthesis.

Notes: This command is used to select or read the polarity of the trigger line for an externally gated input port. If the trigger polarity is set to zero (Ø), the external trigger line must go from low (Øv.) to high (+5v.) to latch the gated port. Conversely, if the trigger polarity is set to one (1), the trigger line must go from high to low. Note, if trigger polarity is set to any value other than zero (Ø), it will equal the least significant bit of the number.

Trigger polarity is set with an equate statement, and may be read like any other I/O.

Examples:

```
DISP TPA1       'Display trigger polarity for the gated digital port in the main module.
TPH3=1          'Set the trigger polarity to one in port #3 in module "H"
TPC(VAR1)=Ø     'Trigger polarity referenced by a numerical expression
```

VAL

Type: String Function

Purpose: Convert an ASCII character string to a equivalent numerical value.

Syntax: VAL ({string expression})

Notes: If the string expression is not representative of a number, the function will return a value of zero (Ø).

The string of the number to be converted may be in any acceptable format - decimal, binary, hexadecimal, or exponential.

As with all functions, if the parameter is a single entity, the parenthesis may be omitted.

Examples:

```
DISP VAL "4.5"       'Display number 4.5
VAR1 = VAL A$ + 23   'Determine numeric equivalent of string A$, add to 23 and
                     'place result in number variable VAR1
```

WAIT

Type: Program Command

Purpose: Causes the program to pause and wait for an external CONT command to proceed.

Syntax: WAIT

Notes: The program will pause at the location of the WAIT command and wait for an external CONT command. When a CONT command is received, the program will continue with the next statement after the WAIT command.

Sources of CONT command: WAIT/CONT Pushbutton (front panel)
"Low" on external WAIT/CONT line (I/O interface)
CONT direct command

Examples: WAIT 'Program halts at this location until an external CONT command is received.

XOR

Type: Logical operator

Purpose: Used in Boolean operations to compare bits. Will cause the logical XOR (Exclusive OR) of two numerical expressions.

Syntax: {numerical expression} XOR {numerical expression}

Examples:

```
VAR1 = 110B              ' 110B = 6
VAR2 = 101B              ' 101B = 5
X = VAR1 XOR VAR2        'Result is 110B XOR 101B = 11B
DISP VAR1 XOR 14         'Display 8 (decimal format) or 1000B
```

Note: XOR can be used in IF/THEN statements to combine comparisons

IF ((PA6 XOR VAR1) = VAR2) OR PA3=5CH THEN GOTO LABEL1

Notes: Logical Operators may operate only on Integers. All floating point numbers will be converted into integers before the logical operation. This truncates all fractional parts of the numbers. Since Integers are always 16 bits in length, any Logical operation will consider all 16 bits, including leading zeros (Ø) and the sign bit. This is very important to consider any time Logical Operators are used.

Logical Operators operate on the "bits" of a number. Because of the way negative numbers are represented in digital computers, the results of a logical operation involving a negative number can appear confusing if the result is shown in "decimal" format (base 10). Viewing the numbers in binary format (base 2) shows each bit of the numbers and the result.

For more on Logical Operators, see Operator Types in the Programming Section.

YES

Type: Operator Input Function

Purpose: Provide direct operator feedback

Syntax: YES

Notes: YES is a numeric function that can have a value of one (1) or zero (Ø), depending on whether or not the YES pushbutton is pressed. YES will equal zero (Ø) if the pushbutton is not pressed, and will equal one (1) if the pushbutton is pressed.

The FT-100a senses the YES pushbutton at the precise moment that the YES function is evaluated. There is no wait or delay. If the YES pushbutton is pressed at any other time, there will be no effect.

This function, along with the NO function, provides a very simple method of operator feedback to simple questions. YES and NO may be used in IF...THEN statements or they may be used in numerical expressions just like any other function. Note, unlike most other functions, YES and NO do <u>not</u> have any parameters.

Examples:

```
IF YES THEN GOTO T1                  'If YES pushbutton is pressed, go to label T1
VAR1=YES+3.4                         'YES can be used like any function
IF YES AND (LA4=1) THEN GOTO L4      'Combine YES with other conditions
```

This routine asks the operator a question, and waits for the result:

```
      DISP "   Is the ": DISP " RED LED on?": DISP "Press YES or NO"
L1:   IF YES THEN BEEP: GOTO L2
      IF NO THEN CALL ERR1: GOTO L3
      GOTO L1
```

The operator is instructed to observe the red LED. If it is on (YES pressed), the FT-100a will beep and go on to label L2. If the LED is not on (NO pressed), the error routine ERR1 is called and routine goes to label L3. The routine sits in a loop, waiting for the YES or NO response.

8.2 Reference - Direct Commands

"Direct commands" are not available as programming commands, but must be enacted through either the keyboard or the serial communications port. Direct commands, like all other commands issued through the keyboard or serial port, must be followed by a carriage return (ENTER or RETURN on most computers and terminals). The syntax descriptions below show a carriage return character (CR) after each command.

CONT

Type: Program Control Command

Purpose: If the program has been "paused" by execution of a WAIT command or WAIT pushbutton being pressed, CONT will cause the program to continue with the next statement.

Syntax: CONT (CR)

Notes: If the program is not in a "pause" state, CONT will have no effect.

DIR (or FILES)

Type: Disk Command

Purpose: Causes a listing of the floppy disk files to be sent out the serial communications line.

Syntax: DIR (CR)

Notes: DIR and FILES are the same command.

DUMP

Type: Disk Command

Purpose: Sends a listing of the "compacted" program out the serial communications port.

Syntax: DUMP (CR)

Notes: The listing is an ASCII representation of the hexadecimal value for each byte of program. There is a header and some overhead information sent along with the program. The compacted form is the most efficient form of program transfer and uses the least amount of space for storage.

Once a listing has started, it may be aborted by pressing any front panel pushbutton.

FILES (or DIR)

Note: FILES and DIR are the same command. See DIR.

FORMAT

Type: Disk Command

Purpose: Formats a floppy disk.

Syntax: FORMAT (CR)

Notes: Before a floppy disk can be used for data or program storage, it must be "formatted". Formatting structures fixed locations on the disk so the disk driver software knows how and where to find information. The FT-100a format is the same as that used by MSDOS compatible computers.

There is a safety check in the format procedure to insure that disks are not unintentionally formatted.

WARNING - Formatting a floppy disk will erase the entire disk, losing all data!

For more information, see the Disk Operation Section.

INFO

Type: System Command

Purpose: Send various system information out the serial communications line, including the current version of firmware, amount of program memory used thus far, remaining memory etc.

Syntax: INFO (CR)

Notes: Following is an explanation of the types of memory:

Bytes used by Program - Number of bytes used to contain the "compacted" form of the program. The maximum program size is 65,535 bytes.

Free RAM - Memory available for use by the program. This does not include any of the memory used for overhead or operating system housekeeping. The maximum amount of free RAM available for use by the program is 65,535 bytes. Free RAM is used primarily for variables, arrays, and strings.

LIST

Type: System Command

Purpose: Sends an ASCII listing of the program out the serial port.

Syntax: LIST {lines to be listed} (CR)

{lines to be listed} - (optional) Beginning and ending line numbers to be listed. Following are the various options for the LIST command:

LIST	List entire program.
LIST {first line#}-	List program starting with {first line#} until end
LIST -{last line#}	List program from beginning until reach {last line#}
LIST {first line#}-{last line#}	List program from {first line#} to {last line#}

Notes: The program will be listed in the ASCII format, allowing editing by a word processor or text editor program. The listing has each program line numbered, beginning with one (1). When editing, it is <u>not</u> important to maintain proper line numbers. When loading a program file back into the FT-100a, all line numbers are eliminated.

Once a listing has started, it may be aborted by pressing any front panel pushbutton.

LOAD

Type: Disk Command

Purpose: Load a program into the FT-100a.

Syntax: LOAD (CR) (Loading program via serial port)

or: LOAD {program name} (CR) (Loading program from floppy disk)

{program name} - Name of program to load from disk. Program name must be in quotes unless it is a string expression.

Notes: If the program name is included in the LOAD command, the program file will be retrieved from the floppy disk. If there is no filename specified, the FT-100a will wait for the program file to be transferred in through the serial communications port.

<u>Floppy Disk Load</u> - The program filename should be complete (including extensions), and must be within quotes. If the extension (".PRG" or ".ASC") is omitted, ".PRG" will be assumed. If the program name is a string variable, quotes must not be used. If the program file is not on the floppy disk, an error will result.

<u>Serial Port Load</u> - ASCII and "compacted" programs use the same command. The FT-100a can identify the format type and handle it accordingly. Note, echo is automatically turned <u>off</u> during serial port program loading, and turned back on (if previously enabled) after loading. If the RESET pushbutton is pressed, or power turned off during program load, echo will be off and will stay off! The COM command or System Set-Up must be used to turn echo back on.

RESET

Type: System Command

Purpose: Performs a "soft" reset of entire system

Syntax: RESET (CR)

Notes: This command performs a "soft" reset, which is similar to pressing the rear panel RESET pushbutton. The RESET pushbutton performs a "hard" reset as well. Most of the capabilities of the two are the same, but some of the internal hardware may require a "hard" reset in order to guarantee proper initialization. A "soft" reset causes the FT-100a to go into its boot-up routine. This is the same routine that the FT-100a goes through each time power is first turned on.

RESET will not erase the program memory, but will stop a program and erase any data that may be in the FT-100a that hasn't been saved to a floppy disk.

Resets are usually only needed when a catastrophic event causes the FT-100a to perform unpredictably, such as a static discharge, etc. Reset can be used to "get out" of a particular routine, or to stop all functioning, but it is not recommended for this purpose. The FT-100a's internal pointers and parameters could be at transitional states and lead to some unpredictable results. Remember, all unsaved data is lost during a reset!

> * * WARNING * *
> If a RESET command is issued while an <u>output</u> disk file is "open", the file and its data could be destroyed!

RUN, START

Type: Program Control Command

Purpose: Immediately starts running the program in the FT-100a.

Syntax: RUN (CR)

or: START (CR) (START and RUN are the same command)

Notes: This command performs the same function as pressing the front panel START pushbutton. The installed program will immediately begin running. Note, all I/O outputs (analog and digital) will be automatically set to zero (Ø), and open data files closed, any time a program is started.

SAVE

Type: Disk Command

Purpose: Save the current program to the floppy disk

Syntax: SAVE (CR) (Save in compacted format)

or: SAVE ,A (CR) (Save in ASCII format)

The SAVE command is used to save an ASCII or compacted program to a floppy disk. If the extra parameter, ",A" is included, the program will be saved in the ASCII format.

Notes: If the program is saved in the ASCII format, the disk may be read by an MSDOS compatible computer. The program may then be edited using a word processor or text editor, and loaded back into the FT-100a.

STOP

Type: Command

Purpose: Immediately stops a running program

Syntax: STOP (CR)

Notes: If a program is running, this command will cause the program to stop at its current location. Once stopped, there is no way for the program to continue from where it was stopped.

All "open" disk data files will be automatically "closed" whenever a STOP command is issued.

Note, all I/O lines will <u>remain</u> at their last state after a STOP command.

WAIT

Type: Program Control Command

Purpose: Causes a running program to "pause" <u>after</u> the current command or statement.

Syntax: WAIT (CR)

Notes: This command performs the same function as pressing the front panel WAIT pushbutton. If a program is running, WAIT will cause the FT-100a to "beep" and the program will pause after the current command or statement. The program will stay in this state until a CONT command is issued as a direct command or front panel pushbutton. When a CONT command is received, the program will continue with the next statement. A STOP command can override the WAIT command.

9.0 Appendix

The following are contained in the Appendices:

APPENDIX A DEFINITIONS

Analog Voltage Range - This is the voltage range that the analog I/O can safely measure or set voltages. The FT-100a has a selectable analog voltage range. Attempts to use values outside this range will result in errors.

Boolean operator - (Also referred to as Logical Operator) Operator which operates on two entities at the bit level. Boolean operators: AND, OR, NOT, NAND, NOR, XOR

Comparison operator - Operators used to compare two expressions.
The following are valid comparison operators:

> Greater than
< Less than
= Equal
>= Greater than or equal
<= Less than or equal
<> Not equal

Direct Mode - Issuing commands and receiving data through the serial communications port. In the "direct mode", commands are executed immediately. This mode of operation allows a terminal or computer to control the FT-100a in real-time.

Extension - In disk filenames, the characters to the right of the decimal point. All filenames do not need extensions. FT-100a program filenames <u>must</u> have extensions ".PRG" or ".ASC".

I/O reference - Any reference to a specific input or output line or port.
Examples: AA3, PA4, LC36, LSB2

I/O Line - Individual input or output line.

Module - Optional input/output expansion units that may be attached to the FT-100a. There may be a maximum of seven external modules connected to the FT-100a at any given time. Each expansion module provides additional input and output capabilities. Modules are referenced by letters A - H, with the FT-100a main unit always being module A.

Null String - An ASCII character string which contains no characters.

Numerical expression - Any expression that reduces to a number.

Port - A group of 8 input/output lines that may be handled together as one entity or separately as individual lines. The structure of a port is such that there is always a least significant bit and a most significant bit, and this structure cannot change.

Serial Communications Port (Serial Port) - RS-232 Serial data port on back of the FT-100a. All communications are done through this port. The FT-100a may be controlled through the serial port as well.

String - Set of alphanumeric characters, such as letters, numbers, symbols, etc. Also known as "ASCII string" or "Character string".
Examples: "ABCDEFG", "Dog", "Result", "@#$%^&*@#$!"
When string data is input, it should be enclosed within quotes.

String expression - Any expression that, when evaluated, reduces to an ASCII string.

APPENDIX B KEYWORDS

Keywords are character strings that have specific meaning to the FT-100a and may not be used for any other purposes. Following is a list of all the Keywords and a brief description of the type:

Input/output references:

AA, AB, AC, AD, AE, AF, AG, AH, LSA, LSB, LSC, LSD, LSE, LSF, LSG, LSH
LA, LB, LC, LD, LE, LF, LG, LH, LRA, LRB, LRC, LRD, LRE, LRF, LRG, LRH
PA, PB, PC, PD, PE, PF, PG, PH, TPA, TPB, TPC, TPD, TPE, TPF, TPG, TPH

Program Commands/Statements:

CLR, FOR, LOOP, NEXT, ELOOP, CALL, RET, RETURN, INC, DEC, INV, DELAY, IN, CLS, IF, BEEP, SUPPLY, WAIT, END, PRINT, DISP, COM, GOTO, SHL, SHR, DINT, EINT, CURSOR, POKE, DIM, READ, DATA, SYS, SS, OPEN, CLOSE, STORE, INPUT, DEL, INX, OUTX, POKEIO, PEEKIO, ANALOG, ON, INKEY, GETKEY

Functions:

ABS, INT, NEG, NOT, PEEK, SIN, COS, TAN, SQR, EOF, RND, YES, NO, VAL, LEN, ASC, TAB, SPC, STR$, CHR$, TIME$, DATE$, LEFT$, RIGHT$, MID$

Misc:

AND, OR, XOR, NAND, NOR, TO, THEN

Direct Commands:

START, RUN, STOP, WAIT, CONT, RESET, LIST, DUMP, DIR, FILES, SAVE, LOAD, FORMAT, INFO

Alphabetical Listing of Keywords:

AA	DATE$	INC	LRD	OUTX	SHR	VAL
AB	DEC	INFO	LRE	PA	SIN	WAIT
ABS	DEL	INKEY	LRF	PB	SPC	XOR
AC	DELAY	INPUT	LRG	PC	SQR	YES
AD	DIM	INV	LRH	PD	SS	
AE	DINT	INT	LSA	PE	STORE	
AF	DIR	INX	LSB	PEEK	STR$	
AG	DISP	LA	LSC		SUPPLY	
AH	DUMP	LB	LSD	PEEKIO	SYS	
ANALOG	EINT	LC	LSE	PF	TAB	
AND	ELOOP	LD	LSF	PG	TAN	
ASC	END	LE	LSG	PH	THEN	
BEEP	EOF	LEN	LSH	POKE	TIME$	
CALL	FILES	LF	MID$	POKEIO	TO	
CHR$	FOR	LG	NAND	PRINT	TPA	
CLOSE	FORMAT	LH	NEG	READ	TPB	
CLR	GET	LIST	NEXT	RESET	TPC	
CLS		LOAD	NO	RET	TPD	
COM	GETKEY	LOOP	NOR	RETURN	TPE	
COS	GOTO	LRA	NOT	RND	TPF	
CURSOR	IF	LRB	OPEN	SAVE	TPG	
DATA	IN	LRC	OR	SHL	TPH	

APPENDIX C SPECIFICATIONS

Notice: These specifications are believed to be accurate, however, Y-tek assumes no responsibility for errors, or their consequences, and reserves the right to make changes at any time without notice.

Front Panel

POWER - (lighted rocker switch) Turn main power on and off.
START/STOP - (momentary pushbutton) Start and stop program
WAIT/CONT - (momentary pushbutton) "Pause" a running program, or continue after a WAIT.
COMMAND - (momentary pushbutton) Access extra commands (disk commands, etc.)
ESC - (momentary pushbutton) "Escape" from some routines.
DISPLAY - 4 Rows x 16 Columns SuperTwist Liquid Crystal Display (LCD) with Electroluminescent Backlighting
DISK DRIVE - 3.5 inch, 1.44M High Density Floppy Disk, MSDOS/IBM PC Compatible.

I/O Interface

CONNECTORS - Latched flat cable IDC headers: 50 pin, 40 pin, 30 pin
(IDC = Insulation Displacement Connector)

INTERCONNECT CABLES - 30 inches long flat cable with strain relief connectors.

USER AVAILABLE LINES:

Digital Input Lines - Five 8-bit ports, addressed as ports or individual lines.

Externally Gated Digital Ports - One of the Digital Input Ports is gated by an external trigger line. Trigger polarity is programmable.

Digital Output Lines - Five 8-bit ports, addressed as ports or individual lines.

Analog Input Lines - Eight analog input lines, ≈2.44 mv. resolution.

Analog Output Lines - Seven analog output lines
Four lines @ ≈2.44 mv. Resolution & three lines @ ≈39.1 mv. resolution.

Power supplies: +5 volts @ 1 amp.
+12 volts @ 1 amp.
-12 volts @ 250 ma.
Ø-10 v. variable voltage supply @ 1 amp.

System Control Lines: START/STOP
WAIT/CONT
TRIG (external trigger line)
RESET

Ground Lines: 10 DC Power and Signal Ground lines
4 Analog Signal Ground lines

HARDWARE SPECIFICATIONS FOR USER I/O INTERFACE

PARAMETER		MIN.	TYP.	MAX.	UNITS	NOTES
<u>DIGITAL INPUTS</u>						
Maximum Input:	Voltage	-35		50	volts	notes 1,8
	Current			+/-20	ma.	notes 1, 5
Recommended Input		0		5.0	volts	
High Level Input		2.0			volts	note 2
Low Level Input				0.8	volts	"
Input Resistance		44650		49350	ohms	47K pullup, note 3
Input Operating Current						
High Level				1	µa.	
Low Level				89.3	µa.	note 3
Input Capacitance			8	12	pf.	
<u>DIGITAL OUTPUTS</u>						
Maximum Voltage Applied to Output		-4.9		9.9	volts	notes 1, 4
Maximum Clamp Current				+/-20	ma.	notes 1, 5
Output Voltage						
High Level (-20 µA load)		4.4		4.5	volts	note 4
(-6 ma. load)		2.86		2.99	volts	"
Low Level (20 µA load)			0.0044	0.1044	volts	"
(6 ma. load)			0.1744	0.2644	volts	"
Output Current						
Maximum per line				+/-35	ma.	
Total for 8 lines of a port				+/-70	ma.	
<u>ANALOG INPUTS</u>						note 6
Maximum Voltage at Input				+/-66	volts	note 1
Input Voltage Range:	0-10v Range	0		10	volts	Recommended Range
	+/-5 v Range	-5		5	volts	Recommended Range
Input Current			45	300	nA.	
Overall Accuracy (+/-)		2.44		1 9.54	mv.	note 7
Input Capacitance			6	45	pf.	
<u>ANALOG OUTPUTS</u>						note 6
Output Voltage Range:	0-10v Range	0		10	volts	
	+/-5v Range	-5		5		
Output Current	Source		20		ma.	note 1
	Sink		10		ma.	note 1
Overall Accuracy (+/-)	Outputs 1-4	2.44		14.54	mv.	note 7
	Outputs 5-8	39.1		136.7		
<u>EXTERNAL CONTROL LINES</u> (Trigger, START/STOP, WAIT/CONT, RESET)						
Maximum Input:	Voltage	-0.5		5.5	volts	note 1
	Current			+/-20	ma.	notes 1, 5
Recommended Input		0		5.0	volts	
High Level Input		2.0			volts	
Low Level Input				0.8	volts	
Input Resistance		9500		10500	ohms	10K pullup to +5v.
Input Operating Current						
High Level				0.001	ma.	
Low Level				0.442	ma.	
Input Capacitance			8	12	pf.	

(See Notes on following page)

NOTES:
1. Absolute maximum ratings. Prolonged operation at or above these ratings may cause damage.
2. The standard input device is HTCMOS (TTL levels). CMOS input level sensing is available as an option. Contact the factory.
3. Each digital input line has a 47K ohm (+/-5%) pullup resistor to +5 v. This value can be changed as an option. Contact the factory.
4. Each digital output line has a 220 ohm current limiting series resistor. This value can be specified or omitted as an option. Contact the factory.
5. Clamp current is the current drawn through the internal clamp diodes.
6. If different analog input/output specifications must be met, contact the factory.
7. Cumulative error, including: accuracy, non-linearity, offset, and gain.
8. Overvoltage ratings are given for one line of a port (8 bits). If all 8 lines of the port are held continuously at maximum overvoltage, the levels are +20/-8 volts.

POWER SUPPLIES AVAILABLE THROUGH I/O INTERFACE

VOLTAGE	MAXIMUM CURRENT (1)	REGULATION	RIPPLE
+ 5 volts	1.0 amp	+/- 2%	50 mv.
+ 12 volts	1.0 amp (2)	+/- 5%	120 mv.
- 12 volts	250 ma.	+/- 5%	120 mv.
0 - 10 volts	1.0 amp (2)	+/- 5%	100 mv.

Notes: 1. Nominal values. All power supply lines have thermal overload protection. This limit may be exceeded for brief periods without damage.

2. This value may derate somewhat if both the +12 v. and the variable supply are operated at maximum current levels for extended periods of time.

Physical Specifications

Power Requirements	100-240 VAC, 47-63 Hz.
Operating Temperature	0° - 50° C
Storage Temperature	-30° to 70° C
Dimensions	11.2"W x 3.6"H x 8.8"D
Unit Weight	5.75 pounds
Shipping Weight	10 pounds

Standard Accessories

1 - YES-NO pod (with 6 ft. cable) (PN 001-0100-00)
1 - 30 inch 50-conductor interconnect cable (PN 803-0000-04)
1 - 30 inch 40-conductor interconnect cable (PN 803-0000-05)
1 - 30 inch 30-conductor interconnect cable (PN 803-0000-06)
1 - AC Power cord, 6 ft. (PN 803-0001-00)
1 - Operators Manual (PN 983-0100-00)

Note: These specifications are believed to be accurate, however, Y-tek assumes no responsibility for errors, or their consequences, and reserves the right to make changes at any time without notice.

APPENDIX D ERROR CODES

When the FT-100a detects an error (programming or operational), the display will indicate an error number and a very brief description. If the error occurs during program execution, the program line number is displayed along with the first 31 characters of the program line. If the error is the result of a direct command issued through the RS-232 input, the error message will also be sent out the serial communications line.

When an error conditions is detected, all "open" disk files will be "closed" (unless the error is a disk related error).

This appendix contains a listing of all errors with a complete description of each. Also included are hints and suggestions for correcting the error.

ERROR: 0 RAM TEST ERROR

This error can only occur during the power-up self-test routine. The FT-100a has detected a read/write error in its RAM memory. Press RESET or turn power off and back on to go through the test again. Sometimes power "glitches" or static discharge can cause an erroneous failure. If the FT-100a continues to fail the RAM TEST, it must be serviced.

ERROR: 1 SYNTAX

This is, by far, the most common error. The FT-100a cannot interpret the command or statement. Misspelling, missing spaces or delimiter characters, etc. can cause this error. Consult Programming Commands in the Reference Section and carefully check the statement.

ERROR: 2 NUMBER OVERFLOW

The FT-100a can handle number values from about (+/-)9.999E-38 to (+/-)3.4E+38. If a number or an internal calculation goes outside this range, a NUMBER OVERFLOW error will result. Sometimes the cause of this error is not obvious. Most of the mathematical routines, and several others as well, perform many mathematical routines that are "invisible" to the operator. If an overflow condition occurs during this type of operation, the error will be indicated.

When the error occurs, use direct mode and "look" at the values of any variables that might be the source of the problem. If all else fails, try performing the same mathematics using some different methods.

ERROR: 3 DEV. NOT PRESENT

An attempt has been made to access an external device that is not present. Depending on the type of device, check to see that all connections are proper and the device is addressed correctly. Consult the manual that came with the device.

ERROR: 4 DIV BY ZERO

The FT-100a cannot process mathematical division with a denominator of zero (Ø). If the source of the error is not obvious, keep in mind that some commands and statements require internal mathematics and if a division by zero is attempted there, the error will result. Sometimes, if numbers are extremely small, after some processing and computer "roundoff", a value may be rounded or truncated to zero and cause this error.

ERROR: 5 FOR w/o NEXT

(FOR without NEXT) - The program has a FOR or LOOP statement but has no corresponding NEXT or ELOOP statement to mark the end of the loop. This error does not show up at the FOR (LOOP) statement, but rather at the end of the program. That is because the program doesn't realize the NEXT statement is missing until the end of the program.

One common cause of this problem, other than the obvious omission of the NEXT statement, is a conditional statement inside a FOR-NEXT loop that causes a jump outside the loop. If conditional statements (IF... THEN...) are used inside a loop, be careful that they do no cause a jump outside the loop.

Go through the program, looking for FOR commands, and make sure each one has a NEXT statement. Be sure the NEXT statement is not "hidden" such that the program does not get to it. If unable to solve the problem, putting the program in single-step mode might help.

ERROR: 6 OUT OF MEMORY

The FT-100a has a fixed amount of RAM memory that must be rationed for many different uses. As a use for the RAM is encountered, the needed memory is "blocked" off and is unavailable for other uses. When there is no more memory to use, the OUT OF MEMORY error will result.

Some of the uses for RAM memory are: Variables (including numerical, string, and arrays), Labels, Program, Operating System, etc..

Arrays use a very large amount of memory. If you are using large arrays, it is best to dimension them close to the actual needed size (See DIM in Reference Section).

Note: When an ASCII program is loaded in through the serial communications port or the disk drive, all comments are eliminated and do not require any memory usage. This is also true of all leading spaces and/or TAB's.

ERROR: 7 RET w/o CALL

The program has encountered a RETURN (or RET) command, but there was no previous CALL command. Each time a CALL command is executed, the location of the CALLing statement is saved. When the program encounters a RETURN command, the program will immediately go to the saved location. If there is no location saved, the error will result.

This error often occurs when the program jumps into a subroutine (using GOTO), or if the subroutines are located at the end of a program that has no END statement. This allows the program to simply continue on into the subroutine as if it were part of the main program.

ERROR: 8 UNDEFINED LABEL

A label has been referenced that is not present in the program. It is acceptable to have lines with labels that are never referenced, but when a label is referenced in a command, but is not present, the error will result. Be very careful of spelling. Also, be sure the label is not in a comment.

ERROR: 9 NO PROGRAM

An attempt has been made to perform a task using the program, but there is not a valid program in the FT-100a. Since the FT-100a's RAM memory is battery-backed, there is usually an installed program. If the START pushbutton is pressed and held during power-up, the FT-100a will go into a setup routine. This routine allows a "system initialization" which, among other things, will erase the program. If a program LOAD routine is aborted (such as pressing the RESET pushbutton), the existing program will have been destroyed. Once a program is destroyed, there is no way of recovering the program. A new program must be loaded in.

ERROR: 10 TOO MANY CHARS

(Too Many Characters) - There are several places where the number of characters must be limited. For instance, variables are limited to 8 characters, labels can be no more than 6 characters, etc. Finding the problem should require no more than a good examination of all labels and variables.

Attempting to set a variable or output to a value (either numerical or string) that is longer than supported can often cause this error.

ERROR: 11 DUPLICATE LABEL

A label has been assigned to more than one program line. This error is normally caught during the process to "compact" an ASCII program. A label can be referenced in multiple program lines, but a given label may be assigned to only one program line.

ERROR: 12 MODULE NOT INSTALLED

This error occurs when an attempt is made to access an external module which is not present. The FT-100a can sense the presence of external modules and read their configuration. If the referenced module is connected to the FT-100a, check the connections and cable for problems. Be sure the module designator code is set correctly for the given module. See the manual for the external module for the module designator code. If an external module is connected after power is turned on, the FT-100a doesn't know it is there.

ERROR: 13 UNMATCHED PAREN.

(Unmatched Parenthesis) - All parenthesis used in numerical and/or string expressions must be *closed*. That is, every opening parenthesis "(" must have a corresponding closing parenthesis ")".

For example: VAR1 = ((VAR2 + 4.5) / 6.78

This is an error! There are two left parenthesis but only one right

ERROR: 14 IMPROPER I/O REF

A reference has been made to an I/O port, line, or device that does not exist. This can be either a module that is not installed or a port or line that is not present in a given module.

If you are sure the reference is correct, check to see that the external module interconnect cable and connectors are properly connected and in working order. If a computer or terminal is connected to the serial port, use the direct mode to access the input/output lines of the module. If the referenced module is external, and you are unable to access it in the direct mode, check that the module designator code is correct (see external module manual).

ERROR: 15 LOOP NESTING

The FT-100a allows up to 8 FOR/NEXT loops to be embedded within each other at any one time. This means loops may be 8 "deep". A loop is embedded within another loop if the FOR-NEXT statement pair are completely inside another FOR-NEXT statement pair. For example:

```
FOR VAR1=1 TO 5          'These loops are 3 "deep"
  FOR VAR2=3 TO 9
    FOR VAR3=1 TO 100
       .
       .
    NEXT VAR3
  NEXT VAR2
NEXT VAR1
```

If an attempt is made to have more than 8 FOR/NEXT loops within each other, the LOOP NESTING error will result.

ERROR: 16 NEXT w/o FOR

(NEXT without FOR) - The program has reached a NEXT (or ELOOP) statement but has no corresponding FOR command. Each and every FOR statement must have a corresponding NEXT statement.

The most obvious cause of this error is an oversight in setting up the FOR-NEXT loops, but there can be other, not so obvious, causes. If a jump command (GOTO or CALL) transfers the program into the middle of a FOR-NEXT loop, this error can result.

Go through the program and pair up the FOR and corresponding NEXT statements. If all else fails, enable single-step and trace through the program.

ERROR: 17 SUBROUTINE NEST

The FT-100a allows subroutines to be "nested" several "deep". That is, subroutines called from within other subroutines. There is a section of RAM memory used for the subroutine "housekeeping", and if the "depth" of the subroutines is too great, there will be insufficient memory space. The actual depth of subroutines allowed depends on several things, but there should be no problem with up to 30 subroutines deep.

If your program has close to 30 subroutines deep, consider a different method of achieving the desired result without using as many nested subroutines.

ERROR: 18 DATA MISMATCH

When an attempt is made to put string data into a numerical entity or vice versa, the DATA MISMATCH error will result. String and numerical data cannot be mixed but often can be converted from one form to the other (using STR$ and VAL functions). Be sure the data is in the correct and expected form.

ERROR: 19 DEVICE TIME-OUT

An attempt has been made to access a device that is not responding. Most often this is the serial communications line, but some external modules or options may also give this error if proper contact cannot be made.

When the FT-100a attempts to send data out the serial communications port but does not receive proper handshaking, it will keep trying for about 10 seconds. If, by the end of this time, no connection has been made, the error will result. If a printer is connected to the serial port, it must be "on-line" or the FT-100a will time-out.

If a computer or terminal is connected to the serial port, and the error occurs, check the connections and cable. Be sure the cable is the correct type (NULL modem), and has the correct connections internally (see Serial Communications in the Appendix). If the device is a printer, be sure it is online.

ERROR: 20 BUFFER OVERFLOW

The working spaces, or "buffers", within the FT-100a allow up to 128 characters at any one time. If an attempt is made to place more than 128 characters in this working space, the error will result. This working space, or buffer area, is used any time information is being input, output, or internally evaluated.

Sometimes it is difficult to know exactly the cause of this error, since it is impossible to "see" inside the FT-100a during the evaluation of an expression. If an expression is getting close to 100 characters (including all punctuation and spaces), try to break up the expression into smaller pieces.

Of course, character strings being input or output cannot be over 128 characters, but these are usually easy to detect. If characters are being input into the serial communications line, and a carriage return character is not detected before 128 characters are input, the error will result.

ERROR: 21 DATA TRANSFER

The format for "compacted" program files and operating system files contains a checksum of the file. The checksum is an 8-bit truncated sum of all the characters in the file. When these files are loaded into the FT-100a through the serial communications port, a new checksum is generated and compared against the checksum in the file being loaded. If the two are not equal, the DATA TRANSFER error results, indicating that somewhere in the loaded file, there is at least one error.

Be sure all communications cables and connectors are secure and in good condition. Try loading the file again. If the error continues, verify that the communications line is functioning by issuing commands in the direct mode. Try loading other program files. If other program files load properly, there is little that can be done. The file itself probably contains errors.

ERROR: 22 OUT OF RANGE

This error occurs when the FT-100a is expecting a number within a given range of values but gets a number out of this range. Some commands will only accept certain values for some parameters. If the input value is not one of the acceptable values, the OUT OF RANGE error will result. For example: COM 4800 The baud rate must be input as one of the acceptable baud rates.

If the cause of the error is not obvious, consult the Programming Commands in the Reference Section.

ERROR: 23 DIMENSION ERROR

This error is caused when either an attempt is made to dimension an array with an "illegal" value, or an insufficiently dimensioned array is referenced. It can also be caused by an attempt to redimension an array.

Dimensioning an array sets up enough memory to store all the array values and must be done if the array dimensions will be over 10 elements. If the array will be no larger than 10 elements in any dimension, the DIM statement can be omitted.

When dimensioning an array (DIM statement), only positive integers greater than zero (Ø) are acceptable. If any other value is used, the DIMENSION ERROR will result.

ERROR: 24 OUT OF DATA

The READ command gets data from the DATA statements that are located within the program. Each time a READ is executed, the next data element is taken from the DATA statements. If a READ is attempted, but all the data has been extracted from the DATA statements, the OUT OF DATA error will result. Remember that data is extracted from the DATA statements sequentially, starting with the first DATA statement in the program. When all the data has been extracted from the first DATA statement, the next DATA statement is used. This continues on for all the READ commands. There must be enough data elements within the DATA statements to cover all READ commands.

Go through the program, matching READ commands with DATA statements. Note that there can be more data than needed and cause no problem, but there must be enough data.

This error can also occur whenever an attempt is made to load in an ASCII program that does not have the "/" end marker (See Program Structure in the Programming Section).

ERROR: 25 NOT DIR COMMAND

(Not a Direct Command) - A command has been issued through the serial communications port that is not a supported command. There are a few commands that are allowable only as programming commands, but not usable in the direct mode. A different command or method must be used to achieve the desired results.

Note: If a direct command has a syntax error, sometimes the error can be reported as a "NOT DIR COMMAND" error.

ERROR: 26 DISK FULL

An attempt has been made to send data or a program file to the floppy disk, and there is not enough room on the floppy disk to hold the information. Files must be deleted on the disk or another disk used.

ERROR: 27 DISK LOAD ERROR

There are many "checks and balances" used in transferring information to and from a floppy disk. These are all internal to the disk operating system and are not accessible to the user. If any of these self-checks fail during a disk data transfer, the DISK LOAD ERROR will result. This error can indicate a problem with the floppy disk itself, or a file that was improperly "closed". If there is a defect on the floppy disk, the only thing that can be done is replace the disk.
If you wish to pursue the problem, and have access to an IBM compatible computer, there are several disk utility programs commercially available that allow thorough disk inspection and some error correction. This is only recommended for the more advanced (and ambitious) programmers. If interested in pursuing this, and unable to find a good disk utility program, contact the factory.

ERROR: 28 FILE TOO LARGE

An attempt has been made to load in a floppy disk program file that is too large to fit into the FT-100a memory. If the program file was saved from the FT-100a there is no question about its being able to "fit". Since programs may be written on a computer and then loaded into the FT-100a through the disk drive or serial communications port, there exists the possibility of having a program too large.

The FT-100a removes all comments, line numbers and leading spaces from each program line as it is loaded in. If a very large program is needed, try to break the program into two or more programs with prompts for the operator to load the next program from the disk drive.

ERROR: 29 OPEN FILE

This error indicates an attempt to "open" a file while one is already open. The FT-100a will allow only <u>one</u> file to be "open" at any given time. Go through the program and make sure any open file is "closed" (CLOSE command) before attempting to open a new file.

ERROR: 30 FILE NOT FOUND

A filename has been referenced that is not on the floppy disk. Be sure all characters of the filename are correct (including the extension), and be sure the correct floppy disk is installed. Use the DIR command (front panel pushbutton or direct command) to view the disk directory.

ERROR: 31 FILE NOT OPEN

An attempt has been made to read (INPUT command) or write (STORE command) information to (or from) a floppy disk file that has not been "opened". For the INPUT and STORE commands to know which file to access, the file must be "opened" prior to any access.

Inspect the program flow and be certain that a proper OPEN command precedes any INPUT or STORE commands.

ERROR: 32 FILE TYPE

This error occurs when a file manipulation is attempted on an incompatible type file. The three-letter extension of the filename defines the type of file. If a manipulation is attempted that is not compatible with the file type, the error will result.

File types: {filename.ASC} - ASCII program file
{filename.PRG} - Compacted program file
{filename.OPR} - Operating system

These extensions may not be used for data files. Data files may use any other extension or no extension at all.

ERROR: 33 INVALID FILENAME

The referenced filename contains some illegal characters. The first character of all filenames must be a letter A - Z. All other characters must be either letters A-Z, underline (_), or dollar sign ($). There can be no more than 8 characters plus 3 extension characters. Be sure all filenames meet these qualifications.

ERROR: 34 DEFECTIVE DISK

A defect has been found in at least one of the floppy disk sectors. During a FORMAT operation, each and every storage location on the floppy disk is tested for proper storage ability. If an error is found, the entire sector is marked as defective (on the disk itself) and is unused. Since the other sectors are operating correctly, the floppy disk is formatted using only the good sectors. If any error is found on a disk, the DEFECTIVE DISK error will be indicated to alert the operator.

Since nearly all floppy disks available today are guaranteed "error free", if an error shows up on a disk, chances are it happened after the disk was purchased. Considering the price of a floppy disk and how much information would be lost if a sector developed an error, it would seem to be good economics to scrap any disk with errors. For this reason, the FT-100a alerts the operator, but the operator has the choice of whether or not to use the defective floppy disk.

Note: If a floppy disk is formatted on another computer, the sectors with errors are not used, but the operator is usually not aware that errors exists. For this reason alone, it is a good idea to format disks in the FT-100a, even though it will probably be slower.

ERROR: 35 DISK READ

Any time the FT-100a reads information from a floppy disk, it reads it one sector at a time. After a sector is read in, it is read in again to compare the two groups of information. If there is any inconsistencies, the process is done again. After several unsuccessful sector reads, the DISK READ error is indicated. The usual cause of this error is due to a floppy disk that has a "marginal" error within the sector. These errors do not show up at all times and for all practical purposes can be considered "intermittent". If the error isn't caught during the format routine, it will be on the disk to cause trouble later.

Oftentimes it is difficult or impossible to recover a file that has a disk error. If the file can be extracted from the disk, the disk should be discarded immediately.

If an attempt is made to access a diskette with an unsupported density, this error can result. The FT-100a supports 1.44M High Density diskettes (DSHD) only.

Since the FT-100a disk format is the same used by IBM PC compatible computers, there are several disk utility programs commercially available that allow thorough disk inspection and some error correction. This is only recommended for the more advanced (and ambitious) programmers. If interested in pursuing this, and unable to find a good disk utility program, contact the factory.

ERROR: 36 DISK WRITE

All information is written to a floppy disk one sector at a time. After each sector of data is written to the disk, it is read back in to verify the information. If there is any discrepancy, the DISK WRITE error results. The usual cause of this error is a defective or improperly formatted floppy disk. A properly formatted disk with this error should be discarded.

If an attempt is made to access a diskette with an unsupported density, this error can result. The FT-100a supports 1.44M High Density diskettes (DSHD) only.

ERROR: 37 WRITE PROTECT

An attempt has been made to store information on a floppy disk that has the "write protect" tab *opened*. All 3.5 inch floppy disks have a small window in a corner. This window has a sliding flap to permit the window to be opened or closed. If the window is open, the disk drive will not allow data to be written to the floppy disk. The window <u>must be closed in order to write</u> to the floppy disk.

ERROR: 38 DRIVE NOT READY

Access to a floppy disk has been attempted, but the disk drive was not ready. Most often this happens when the floppy disk is not installed in the drive or not fully seated and latched. When pressing the floppy disk into the disk drive, be sure the disk latches and the release pushbutton pops out. If the floppy disk appears to be installed correctly, but the error still occurs, try releasing the disk completely and then re-inserting.

APPENDIX E HARDWARE INTERFACING

The hardware interface to the FT-100a is done through the 50, 40 and 30 pin connectors across the bottom of the front panel. The programming language permits an easy and direct interface to the input and output lines.

The pin-out information for the I/O connectors is given in two different formats: first, with reference to the I/O function, then again in order of the connector and pin number.

For the specifications for each of the I/O lines, see Specifications in the Appendix.

I/O DESCRIPTION BY FUNCTION:

SUPPLY LINES

SUPPLY LINE DESCRIPTION	LIMIT CURRENT	CONNECTOR	PIN#
+ 5 VOLTS	1 AMP	40 PIN	20,22
+ 12 VOLTS	1 AMP	40 PIN	24,26
- 12 VOLTS	250 ma.	40 PIN	32
VARIABLE VOLTAGE	1 AMP	40 PIN	28,30
DC GROUND		50 PIN 40 PIN 30 PIN	1,50 6,8,10,12,14,16,18 29
ANALOG GROUND		40 PIN 30 PIN	2,4 12,14

CONTROL LINES

CONTROL FUNCTION	PULLUP RESISTOR VALUE (TO +5V.)	ACTIVE HIGH/LOW	CONNECTOR	PIN#
START/STOP	10 K Ω	LOW	40 PIN	38
WAIT/CONT	10 K Ω	LOW	40 PIN	40
EXTERNAL TRIG	10 K Ω	LOW	40 PIN	34
RESET	10 K Ω	LOW	40 PIN	36

DIGITAL OUTPUT PORTS (& LINES)

PORT# & PORT REFERENCE	BIT#	LINE REFERENCE (note 1)	CONNECTOR	PIN#
PORT# 1, PA1	0	LA1, LA1:1	40 PIN	25
	1	LA2, LA1:2		27
	2	LA3, LA1:3		29
	3	LA4, LA1:4		31
	4	LA5, LA1:5		33
	5	LA6, LA1:6		35
	6	LA7, LA1:7		37
	7	LA8, LA1:8		39
PORT# 2, PA2	0	LA9, LA2:1	30 PIN	16
	1	LA10, LA2:2		18
	2	LA11, LA2:3		20
	3	LA12, LA2:4		22
	4	LA13, LA2:5		24
	5	LA14, LA2:6		26
	6	LA15, LA2:7		28
	7	LA16, LA2:8		30
PORT# 3, PA3	0	LA17, LA3:1	50 PIN	2
	1	LA18, LA3:2		4
	2	LA19, LA3:3		6
	3	LA20, LA3:4		8
	4	LA21, LA3:5		10
	5	LA22, LA3:6		12
	6	LA23, LA3:7		14
	7	LA24, LA3:8		16
PORT# 4, PA4	0	LA25, LA4:1	50 PIN	18
	1	LA26, LA4:2		20
	2	LA27, LA4:3		22
	3	LA28, LA4:4		24
	4	LA29, LA4:5		26
	5	LA30, LA4:6		28
	6	LA31, LA4:7		30
	7	LA32, LA4:8		32
PORT# 5, PA5	0	LA33, LA5:1	50 PIN	34
	1	LA34, LA5:2		36
	2	LA35, LA5:3		38
	3	LA36, LA5:4		40
	4	LA37, LA5:5		42
	5	LA38, LA5:6		44
	6	LA39, LA5:7		46
	7	LA40, LA5:8		48

DIGITAL INPUT PORTS (& LINES)

PORT# & PORT REFERENCE	BIT#	LINE REFERENCE (note 1)	CONNECTOR	PIN#
PORT# 6, PA6	0	LA41, LA6:1	40 PIN	9
Digital	1	LA42, LA6:2		11
I/P	2	LA43, LA6:3		13
	3	LA44, LA6:4		15
	4	LA45, LA6:5		17
	5	LA46, LA6:6		19
	6	LA47, LA6:7		21
	7	LA48, LA6:8		23
PORT# 7, PA7	0	LA49, LA7:1	30 PIN	13
	1	LA50, LA7:2		15
	2	LA51, LA7:3		17
	3	LA52, LA7:4		19
	4	LA53, LA7:5		21
	5	LA54, LA7:6		23
	6	LA55, LA7:7		25
	7	LA56, LA7:8		27
PORT# 8, PA8	0	LA57, LA8:1	50 PIN	3
Digital	1	LA58, LA8:2		5
I/P	2	LA59, LA8:3		7
	3	LA60, LA8:4		9
	4	LA61, LA8:5		11
	5	LA62, LA8:6		13
	6	LA63, LA8:7		15
	7	LA64, LA8:8		17
PORT# 9, PA9	0	LA65, LA9:1	50 PIN	19
	1	LA66, LA9:2		21
	2	LA67, LA9:3		23
	3	LA68, LA9:4		25
	4	LA69, LA9:5		27
	5	LA70, LA9:6		29
	6	LA71, LA9:7		31
	7	LA72, LA9:8		33
PORT# 10, PA10	0	LA73, LA10:1	50 PIN	35
	1	LA74, LA10:2		37
(Note: This port may	2	LA75, LA10:3		39
be triggered externally	3	LA76, LA10:4		41
by the TRIG line)	4	LA77, LA10:5		43
	5	LA78, LA10:6		45
	6	LA79, LA10:7		47
	7	LA80, LA10:8		49

ANALOG INPUT AND OUTPUT LINES

OUTPUT			INPUT		
ANALOG REFERENCE	CONNECTOR	PIN#	ANALOG REFERENCE	CONNECTOR	PIN#
AA1	40 PIN	5	AA9	40 PIN	1
AA2	40 PIN	7	AA10	40 PIN	3
AA3	30 PIN	2	AA11	30 PIN	1
AA4	30 PIN	4	AA12	30 PIN	3
AA5	30 PIN	6	AA13	30 PIN	5
AA6	30 PIN	8	AA14	30 PIN	7
AA7	30 PIN	10	AA15	30 PIN	9
			AA16	30 PIN	11

I/O DESCRIPTION BY CONNECTOR AND PIN

50 PIN CONNECTOR

PIN#	DESCRIPTION	PROGRAM REFERENCE
1	DC GROUND	
2	Bit 0 of Digital OUTPUT Port 3	LA17, LA3:1, or bit 0 of PA3 (note 1)
3	Bit 0 of Digital INPUT Port 8	LA57, LA8:1, or bit 0 of PA8
4	Bit 1 of Digital OUTPUT Port 3	LA18, LA3:2, or bit 1 of PA3
5	Bit 1 of Digital INPUT Port 8	LA58, LA8:2, or bit 1 of PA8
6	Bit 2 of Digital OUTPUT Port 3	LA19, LA3:3, or bit 2 of PA3
7	Bit 2 of Digital INPUT Port 8	LA59, LA8:3, or bit 2 of PA8
8	Bit 3 of Digital OUTPUT Port 3	LA20, LA3:4, or bit 3 of PA3
9	Bit 3 of Digital INPUT Port 8	LA60, LA8:4, or bit 3 of PA8
10	Bit 4 of Digital OUTPUT Port 3	LA21, LA3:5, or bit 4 of PA3
11	Bit 4 of Digital INPUT Port 8	LA61, LA8:5, or bit 4 of PA8
12	Bit 5 of Digital OUTPUT Port 3	LA22, LA3:6, or bit 5 of PA3
13	Bit 5 of Digital INPUT Port 8	LA62, LA8:6, or bit 5 of PA8
14	Bit 6 of Digital OUTPUT Port 3	LA23, LA3:7, or bit 6 of PA3
15	Bit 6 of Digital INPUT Port 8	LA63, LA8:7, or bit 6 of PA8
16	Bit 7 of Digital OUTPUT Port 3	LA24, LA3:8, or bit 7 of PA3
17	Bit 7 of Digital INPUT Port 8	LA64, LA8:8, or bit 7 of PA8
18	Bit 0 of Digital OUTPUT Port 4	LA25, LA4:1, or bit 0 of PA4
19	Bit 0 of Digital INPUT Port 9	LA65, LA9:1, or bit 0 of PA9
20	Bit 1 of Digital OUTPUT Port 4	LA26, LA4:2, or bit 1 of PA4
21	Bit 1 of Digital INPUT Port 9	LA66, LA9:2, or bit 1 of PA9
22	Bit 2 of Digital OUTPUT Port 4	LA27, LA4:3, or bit 2 of PA4
23	Bit 2 of Digital INPUT Port 9	LA67, LA9:3, or bit 2 of PA9
24	Bit 3 of Digital OUTPUT Port 4	LA28, LA4:4, or bit 3 of PA4
25	Bit 3 of Digital INPUT Port 9	LA68, LA9:4, or bit 3 of PA9
26	Bit 4 of Digital OUTPUT Port 4	LA29, LA4:5, or bit 4 of PA4
27	Bit 4 of Digital INPUT Port 9	LA69, LA9:5, or bit 4 of PA9
28	Bit 5 of Digital OUTPUT Port 4	LA30, LA4:6, or bit 5 of PA4
29	Bit 5 of Digital INPUT Port 9	LA70, LA9:6, or bit 5 of PA9
30	Bit 6 of Digital OUTPUT Port 4	LA31, LA4:7, or bit 6 of PA4
31	Bit 6 of Digital INPUT Port 9	LA71, LA9:7, or bit 6 of PA9
32	Bit 7 of Digital OUTPUT Port 4	LA32, LA4:8, or bit 7 of PA4
33	Bit 7 of Digital INPUT Port 9	LA72, LA9:8, or bit 7 of PA9
34	Bit 0 of Digital OUTPUT Port 5	LA33, LA5:1, or bit 0 of PA5
35	Bit 0 of Digital INPUT Port 10	LA73, LA10:1, or bit 0 of PA10
36	Bit 1 of Digital OUTPUT Port 5	LA34, LA5:2, or bit 1 or PA5
37	Bit 1 of Digital INPUT Port 10	LA74, LA10:2, or bit 1 of PA10
38	Bit 2 of Digital OUTPUT Port 5	LA35, LA5:3, or bit 2 or PA5
39	Bit 2 of Digital INPUT Port 10	LA75, LA10:3, or bit 2 of PA10
40	Bit 3 of Digital OUTPUT Port 5	LA36, LA5:4, or bit 3 or PA5
41	Bit 3 of Digital INPUT Port 10	LA76 LA10:4, or bit 3 of PA10
42	Bit 4 of Digital OUTPUT Port 5	LA37, LA5:5, or bit 4 or PA5
43	Bit 4 of Digital INPUT Port 10	LA77, LA10:5, or bit 4 of PA10
44	Bit 5 of Digital OUTPUT Port 5	LA38, LA5:6, or bit 5 or PA5
45	Bit 5 of Digital INPUT Port 10	LA78, LA10:6, or bit 5 of PA10
46	Bit 6 of Digital OUTPUT Port 5	LA39, LA5:7, or bit 6 or PA5
47	Bit 6 of Digital INPUT Port 10	LA79, LA10:7, or bit 6 of PA10
48	Bit 7 of Digital OUTPUT Port 5	LA40, LA5:8, or bit 7 or PA5
49	Bit 7 of Digital INPUT Port 10	LA80, LA10:8, or bit 7 of PA10
50	DC GROUND	

40 PIN CONNECTOR

PIN#	DESCRIPTION	PROGRAM REFERENCE
1	Analog INPUT #9	AA9
2	Analog GROUND	
3	Analog INPUT #10	AA10
4	Analog GROUND	
5	Analog OUTPUT #1	AA1
6	DC GROUND	
7	Analog OUTPUT #2	AA2
8	DC GROUND	
9	Bit 0 of Digital INPUT Port 6	LA41, LA6:1, or bit 0 of PA6 (note 1)
10	DC GROUND	
11	Bit 1 of Digital INPUT Port 6	LA42, LA6:2, or bit 1 of PA6
12	DC GROUND	
13	Bit 2 of Digital INPUT Port 6	LA43, LA6:3, or bit 2 of PA6
14	DC GROUND	
15	Bit 3 of Digital INPUT Port 6	LA44, LA6:4, or bit 3 of PA6
16	DC GROUND	
17	Bit 4 of Digital INPUT Port 6	LA45, LA6:5, or bit 4 of PA6
18	DC GROUND	
19	Bit 5 of Digital INPUT Port 6	LA46, LA6:6, or bit 5 of PA6
20	+ 5 volt SUPPLY (@ 1 amp)	
21	Bit 6 of Digital INPUT Port 6	LA47, LA6:7, or bit 6 of PA6
22	+ 5 volt SUPPLY (@ 1 amp)	
23	Bit 7 of Digital INPUT Port 6	LA48, LA6:8, or bit 7 of PA6
24	+ 12 volt SUPPLY (@ 1 amp)	
25	Bit 0 of Digital OUTPUT Port 1	LA1, LA1:1, or bit 0 of PA1
26	+ 12 volt SUPPLY (@ 1 amp)	
27	Bit 1 of Digital OUTPUT Port 1	LA2, LA1:2, or bit 1 of PA1
28	Variable voltage SUPPLY (@ 1 amp)	SUPPLY
29	Bit 2 of Digital OUTPUT Port 1	LA3, LA1:3, or bit 2 of PA1
30	Variable voltage SUPPLY (@ 1 amp)	SUPPLY
31	Bit 3 of Digital OUTPUT Port 1	LA4, LA1:4, or bit 3 of PA1
32	- 12 volt SUPPLY (@ 1 amp)	
33	Bit 4 of Digital OUTPUT Port 1	LA5, LA1:5, or bit 4 of PA1
34	EXTERNAL TRIGGER INPUT	TPA10, LSA10, LRA10 (10k pullup resistor)
35	Bit 5 of Digital OUTPUT Port 1	LA6, LA1:6, or bit 5 of PA1
36	RESET	Active low (10k pullup resistor)
37	Bit 6 of Digital OUTPUT Port 1	LA7, LA1:7, or bit 6 of PA1
38	START/STOP	Active low, (10k pullup resistor)
39	Bit 7 of Digital OUTPUT Port 1	LA8, LA1:1, or bit 7 of PA1
40	WAIT/CONT	Active low (10k pullup resistor)

30 PIN CONNECTOR

PIN#	DESCRIPTION	PROGRAM REFERENCE
1	Analog INPUT #11	AA11
2	Analog OUTPUT #3	AA3
3	Analog INPUT #12	AA12
4	Analog OUTPUT #4	AA4
5	Analog INPUT #13	AA13
6	Analog OUTPUT #5	AA5
7	Analog INPUT #14	AA14
8	Analog OUTPUT #6	AA6
9	Analog INPUT #15	AA15
10	Analog OUTPUT #7	AA7
11	Analog INPUT #16	AA16
12	Analog GROUND	
13	Bit 0 of Digital INPUT Port 7	LA49, LA7:1 or bit 0 of PA7 (note 1)
14	Analog GROUND	
15	Bit 1 of Digital INPUT Port 7	LA50, LA7:2 or bit 1 of PA7
16	Bit 0 of Digital OUTPUT Port 2	LA9, LA2:1, or bit 0 of PA2
17	Bit 2 of Digital INPUT Port 7	LA51, LA7:3 or bit 2 of PA7
18	Bit 1 of Digital OUTPUT Port 2	LA10, LA2:2, or bit 1 of PA2
19	Bit 3 of Digital INPUT Port 7	LA52, LA7:4 or bit 3 of PA7
20	Bit 2 of Digital OUTPUT Port 2	LA11, LA2:3, or bit 2 of PA2
21	Bit 4 of Digital INPUT Port 7	LA53, LA7:5, or bit 4 of PA7
22	Bit 3 of Digital OUTPUT Port 2	LA12, LA2:4, or bit 3 of PA2
23	Bit 5 of Digital INPUT Port 7	LA54, LA7:6 or bit 5 of PA7
24	Bit 4 of Digital OUTPUT Port 2	LA13, LA2:5, or bit 4 of PA2
25	Bit 6 of Digital INPUT Port 7	LA55, LA7:7, or bit 6 of PA7
26	Bit 5 of Digital OUTPUT Port 2	LA14, LA2:6, or bit 5 of PA2
27	Bit 7 of Digital INPUT Port 7	LA56, LA7:8, or bit 7 of PA7
28	Bit 6 of Digital OUTPUT Port 2	LA15, LA2:7, or bit 6 of PA2
29	DC GROUND	
30	Bit 7 of Digital OUTPUT Port 2	LA16, LA2:8, or bit 7 of PA2

Notes:

1. Digital I/O lines may be referenced as line numbers (LA34), bit# of a port (LA3:5), or 8-bit ports (PA7).

APPENDIX F SERIAL COMMUNICATIONS

The FT-100a uses RS-232C as its communications standard. This appendix contains the information needed to interface with the FT-100a via its serial communications port.

SERIAL COMMUNICATIONS CONNECTOR PIN-OUT

FUNCTION	NAME	DB25 PIN #
Chassis Ground	CG	1
Transmitted Data	TXD	2
Received Data	RXD	3
Request to Send	RTS	4
Clear to Send	CTS	5
Signal Ground	GND	7
Serial Port Select A (note 1)	n/a	9
Serial Port Select B (note 1)	n/a	10

(Note 1 - Lines 9 and 10 are used by the 1041 Serial Port Expander)

WARNING - The FT-100a uses lines 9 and 10 for the serial port expander control. Be sure that any device connected to the serial communications port does not use these lines. If there is any doubt, the interconnecting cable should not have connections on these two lines.

No hardware configuration changes can be made to the serial communications port. The FT-100a always uses RTS/CTS handshaking. If the connecting serial device does not support RTS/CTS handshaking, use a cable that has RTS and CTS shorted together at the FT-100a end (Null-modem cable).

COMMUNICATIONS CABLE

The FT-100a is configured as Data Terminal Equipment (DTE) and evokes RTS/CTS handshaking at all times. Unless the communicating device is a printer or other slow device, handshaking is usually not needed. Most printers will require, and support, handshaking. If handshaking is not required (which is the case for most data terminals and computers), RTS and CTS can be shorted together at the FT-100a end to bypass handshaking. A "null-modem" cable or adapter will accomplish this. A male DB25 connector is required at the FT-100a end of the cable. The other end of the cable should be compatible with the device with which it is to be connected.

12 9

Following are schematics of both versions of the required cable:

5 8

<u>"Null-modem" Cable</u> - Recommended for most data terminals and computers:
(Unless a printer is the serial device, try using this cable first)

```
                                              4    7
              (Male DB25)                   (DB25)  ( DB9)
                 TXD (2) >-----------------< (3)    (2)  RXD
                 RXD (3) >-----------------< (2)    (3)  TXD
(FT-100a)        RTS (4) >-┐                8       1                  (Data Terminal)
                 CTS (5) >-┘                6       6
                 GND (7) >-----------------< (7)    (5)  GND
                                          -< (9)               (Note 1)
                                          -< (10)              (Note 1)
                                             20     4
```

<u>Full Handshaking Cable</u> - Usually required by serial printers and "slow" serial devices:

```
              (Male DB25)                   (DB25)  ( DB9)
                 TXD (2) >-----------------< (3)    (2)  RXD
                 RXD (3) >-----------------< (2)    (3)  TXD
(FT-100a)        RTS (4) >-----------------<                   (Note 2)   (Data Terminal)
                 CTS (5) >-----------------<                   (Note 2)
                 GND (7) >-----------------< (7)    (5)  GND
                                          -< (9)               (Note 1)
                                          -< (10)              (Note 1)
```

Notes:
1. Lines 9 and 10 are used for the serial port expander control. Be sure that any device connected to the serial communications port does not use these lines. If there is any doubt, the interconnecting cable should not have connections on these two lines.

2. The two handshaking lines (RTS & CTS) should be connected to the handshaking lines supported in the communicating device. Often this is RTS & CTS, but some devices support DSR & DTR. Consult the manual for the serial device. Keep in mind that RTS is "driven" by the FT-100a and CTS is "read" by the FT-100a.

<u>OPERATION</u>

When a character is to be sent out the serial communications line, the following sequence is followed:

1. FT-100a pulls RTS line high.

2. CTS line is monitored for a high level. If CTS does not go high within about 10 seconds, a DEVICE TIME-OUT error will result.

3. When CTS goes high, one character is sent out the TXD line. This sequence is repeated until all characters have been sent.

<u>SERIAL DATA FORMAT</u> - The FT-100a conforms to the following serial data protocol:

1 START bit
8 DATA bits
2 STOP bits
No parity

There are several facts concerning the serial port that might be helpful when interfacing:

- Communications is always half-duplex. In other words, data can go only one direction at any given time. If data is being sent, all incoming data is ignored.

- RTS/CTS handshaking is always used for data transmitted out the serial port. There is no handshaking for incoming data.

- The serial input line is connected to a hardware interrupt line in the FT-100a. If interrupts are disabled, data cannot be received. The only exception to this is when the Direct Command Mode is enabled and a GET command is used to input one character.

- The serial input buffer can hold up to 128 characters. If an attempt is made to input more than 128 characters, the BUFFER OVERFLOW error will result. In most cases, when data is read from the serial port, the buffer is cleared. The exception to this occurs when the GET command is used with Direct Command Mode disabled. Only one character is removed from the buffer in this case.

- Any time Direct Command Mode is enabled, and a carriage return (ASCII ØDH) is received, the input character string will be interpreted as a direct command. Also, with Direct Command Mode and Echo enabled, the backspace character (Ø8) will cause the FT-100a to send the following character string: Ø8, 2ØH, Ø8 (backspace, space, backspace).

- The serial input buffer is cleared of all data in the following conditions:
  - Program Starts
  - Program Ends
  - Program is Stopped
  - Error condition
  - Direct Command Mode status is changed (turned on, or turned off)
  - Beginning of the IN command
  - Beginning of the GET command if the Direct Command Mode is enabled
  - Whenever the serial port is switched. Note: This is still true even if a 1041 Serial Port Expander is not connected.

- If the Direct Command Mode is disabled, and there has been incoming data, and a WAIT command is executed (either WAIT pushbutton or program command), the data will remain in the serial input buffer. If an attempt is made to execute a direct command while in the "wait" state, there will most likely be a SYNTAX error, since the data in the buffer will be interpreted as a command.

SERIAL COMMUNICATIONS COMMANDS

Following are the commands that interact with the serial port, and a brief description of each:
(NOTE: See the operators manual for detailed information on these commands)

PRINT - Sends character strings out the serial port. While characters are being transmitted, all incoming data is ignored until the transmission is finished. If the FT-100a senses incoming data at the end of the transmission, it will immediately begin receiving. This can cause erroneous results if the FT-100a begins receiving in the middle of a character.

IN - Clears the serial input buffer and waits for a character string ending with a carriage return (ASCII ØDH). Interrupts are not disabled during the IN command, but the Direct Command Mode is disabled.

GET - Gets one character from the serial port. The GET command functions differently, depending on whether or not the Direct Command Mode is enabled.

Direct Command Mode Enabled

When the GET command begins, the input buffer is cleared. The program then waits for one character to be input. This character is placed in the referenced string variable. If no character is received, the only way to get the program out of the GET command is to press the RESET pushbutton. For this reason, it is usually advisable to disable the Direct Command Mode when using the GET command to input characters.

Direct Command Mode Disabled

All serial data is placed in the serial input buffer in a first-in, first-out fashion. The buffer can hold up to 128 characters at any given time. The GET command will "get" the first character and place it in the referenced string variable. Interrupts must be enabled for data to be received into the buffer.

If the Direct Command Mode is disabled, and a WAIT command is issued (WAIT pushbutton or program command), there will likely be characters in the input buffer. If attempt is made to issue a direct command, a SYNTAX error will usually result.

TWO-WAY COMMUNICATIONS

One of the safest methods to use when two devices must communicate with each other is to use a form of software handshaking. This can usually be as simple as placing a never-used character on the end of the data string to signal to the receiving device that it may begin sending data. Two FT-100a's can easily communicate with each other using this method.

Most serial communications troubles are caused by timing problems. This is especially true when using a half-duplex system with no handshaking. Be sure to allow enough time for the receiving end to process its data. Also, be aware of the contents of the input buffer at all times. This also includes knowing when the buffer is cleared, and when it is not.

APPENDIX G KEYBOARD OPERATION

The FT-100a has provisions for attachment of a standard PC type keyboard. The keyboard can be used as an input device for issuance of Direct Commands or for program data input. Following is information relevant to the use of the keyboard:

- The keyboard input is connected to a hardware interrupt line in the FT-100a. If interrupts are disabled, the keyboard cannot be used to input data. The only exception to this is when the Direct Command Mode is enabled and a GETKEY command is used to input one character.
- The keyboard protocol used in the FT-100a is commonly known as the AT Keyboard Protocol.
- Each key has a unique keycode output. Standard ASCII codes are used for the alpha/numeric characters. Consult the keycode listing for all the output codes.
- The GETKEY command performs differently depending on whether the Direct Command Mode is enabled. See GETKEY in the REFERENCE section for details.
- When Direct Command Mode is enabled and a carriage return (ASCII ØDH) is received, the input character string will be interpreted as a direct command.
- The keyboard input buffer can hold up to 128 characters. If an attempt is made to input more than 128 characters, the BUFFER OVERFLOW error will result. In most cases, when data is read from the keyboard input, the buffer is cleared. The exception to this occurs when the GETKEY command is used with Direct Command Mode disabled. Only one character is removed from the buffer in this case.
- The keyboard input buffer is cleared of all data in the following conditions:
  - Program Starts
  - Program Ends
  - Program is Stopped
  - Error condition
  - Direct Command Mode status is changed (turned on, or turned off)
  - Beginning of the INKEY command
  - Beginning of the GETKEY command if the Direct Command Mode is enabled

Following is a brief program segment that could be used to input a serial number in a test program:

```
        -
        -
CLS: DISP "Input Ser#"     'Instruct the operator to input serial number
INKEY SER$                 'Wait for operator to input character string for serial number
SER = VAL (SER$)           'Convert string into number.
STORE SER                  'Save serial number in disk file
        -
        -
```

Whenever a key is pressed on the optional keyboard, the FT-100a decodes the incoming keycode into a more usable code. This code is the standard ASCII code whenever possible, however, there are several keys that are not normally recognized by the ASCII format. The actual character codes generated by the FT-100a are given below. (Noté: Hexadecimal value is given in parenthesis).

KEY	CODE	
{backspace}	8	(08h)
TAB	9	(09h)
{return}	13	(0Dh)
ESC	27	(1Bh)
{space}	32	(20h)
!	33	(21h)
"	34	(22h)
#	35	(23h)
$	36	(24h)
%	37	(25h)
&	38	(26h)
'	39	(27h)
(	40	(28h)
)	41	(29h)
*	42	(2Ah)
+	43	(2Bh)
,	44	(2Ch)
-	45	(2Dh)
.	46	(2Eh)
/	47	(2Fh)
0	48	(30h)
1	49	(31h)
2	50	(32h)
3	51	(33h)
4	52	(34h)
5	53	(35h)
6	54	(36h)
7	55	(37h)
8	56	(38h)
9	57	(39h)
:	58	(3Ah)
;	59	(3Bh)
<	60	(3Ch)
=	61	(3Dh)
>	62	(3Eh)
?	63	(3Fh)
@	64	(40h)
A	65	(41h)
B	66	(42h)
C	67	(43h)
D	68	(44h)
E	69	(45h)
F	70	(46h)
G	71	(47h)
H	72	(48h)
I	73	(49h)
J	74	(4Ah)
K	75	(4Bh)
L	76	(4Ch)
M	77	(4Dh)

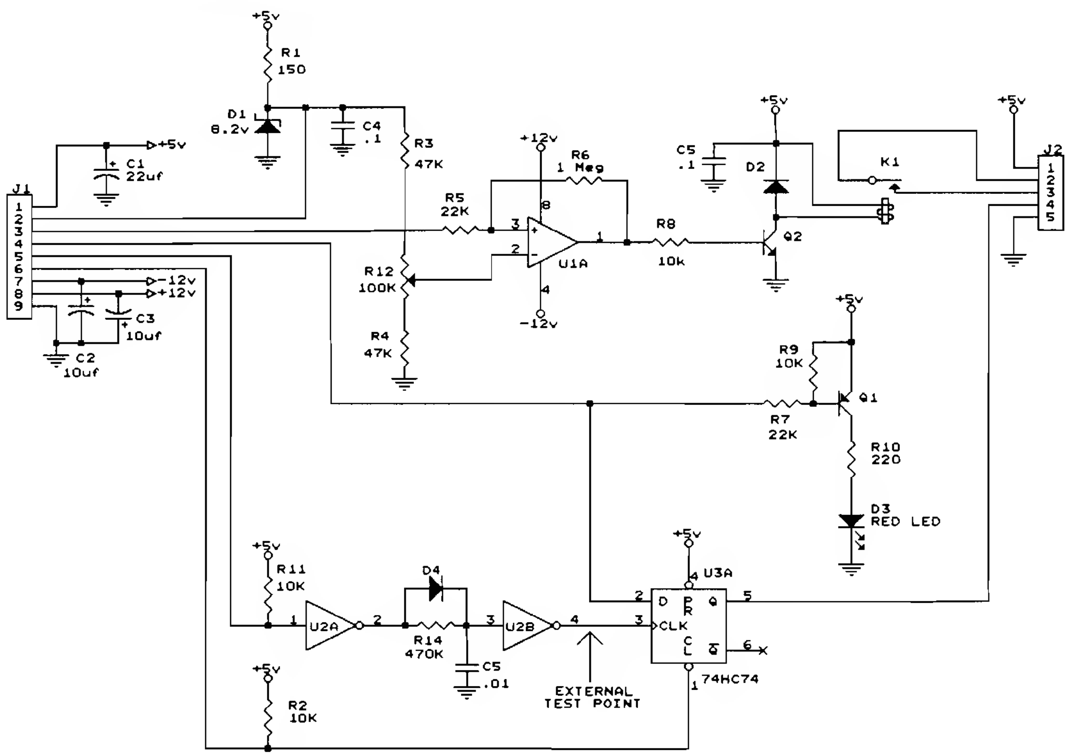
KEY	CODE	
N	78	(4Eh)
O	79	(4Fh)
P	80	(50h)
Q	81	(51h)
R	82	(52h)
S	83	(53h)
T	84	(54h)
U	85	(55h)
V	86	(56h)
W	87	(57h)
X	88	(58h)
Y	89	(59h)
Z	90	(5Ah)
[	91	(5Bh)
\	92	(5Ch)
]	93	(5Dh)
^	94	(5Eh)
_	95	(5Fh)
`	96	(60h)
a	97	(61h)
b	98	(62h)
c	99	(63h)
d	100	(64h)
e	101	(65h)
f	102	(66h)
g	103	(67h)
h	104	(68h)
i	105	(69h)
j	106	(6Ah)
k	107	(6Bh)
l	108	(6Ch)
m	109	(6Dh)
n	110	(6Eh)
o	111	(6Ff)
p	112	(70h)
q	113	(71h)
r	114	(72h)
s	115	(73h)
t	116	(74h)
u	117	(75h)
v	118	(76h)
w	119	(77h)
x	120	(78h)
y	121	(79h)
z	122	(7Ah)
{	123	(7Bh)
|	124	(7Ch)
}	125	(7Dh)
~	126	(7Eh)
DEL	127	(7Fh)

KEY	CODE	
{cursor right}	192	(C0h)
{cursor left}	193	(C1h)
{cursor up}	194	(C2h)
{cursor down}	195	(C3h)
PRINT SCREEN	196	(C4h)
SCROLL LOCK	197	(C5h)
PAUSE	198	(C6h)
INSERT	199	(C7h)
HOME	200	(C8h)
PAGE UP	201	(C9h)
PAGE DOWN	202	(CAh)
END	203	(CBh)
Windows (left)	205	(CDh)
Windows (right)	206	(CEh)
Windows {pointer}	207	(CFh)
F1 {shift}	225	(E1h)
F2 {shift}	226	(E2h)
F3 {shift}	227	(E3h)
F4 {shift}	228	(E4h)
F5 {shift}	229	(E5h)
F6 {shift}	230	(E6h)
F7 {shift}	231	(E7h)
F8 {shift}	232	(E8h)
F9 {shift}	233	(E9h)
F10 {shift}	234	(EAh)
F11 {shift}	235	(EBh)
F12 {shift}	236	(ECh)
F1	241	(F1h)
F2	242	(F2h)
F3	243	(F3h)
F4	244	(F4h)
F5	245	(F5h)
F6	246	(F6h)
F7	247	(F7h)
F8	248	(F8h)
F9	249	(F9h)
F10	250	(FAh)
F11	251	(FBh)
F12	252	(FCh)

APPENDIX H EXAMPLE - TEST CONFIGURATION

This is an example of a test configuration and program that might be used to test a simple PC board. The circuit is fictitious but hopefully it will show some of the capabilities of the FT-100a and help get you started.

Schematic of PC board under test



Circuit Description

This circuit senses a voltage change across a thermistor, and switches a relay when the temperature reaches a programmable point. The switch-point is adjustable, and the test operator is to adjust the threshold for a given temperature. For this configuration, the operator must adjust R12 for the relay to switch at 3.8 volts.

There is some logic on this board as well. Connector J1 contains all the input lines for the logic. If there is a low on pin 4 of J1, LED D3 will light. This same line feeds the "D" input of the flip-flop U3. Whenever the clock (pin 3, U3) goes high, the value at "D" will be transferred to the "Q" output (pin 4, J2). Pin 6 of J1 is used to reset the flip-flop.

When pin 5 of J1 goes high, the voltage across capacitor C5 will slowly fall. When this voltage reaches the threshold of inverter U2, the output will go high and the flip-flop will latch the value at "D" onto "Q".

Interface Configuration

In order to test this PC board, an interface fixture must be built. Often this interface can be a simple cable, or it may be a "bed-of-nails" type fixture. It depends on the mechanical configuration of the unit to be tested, as well as the extent of testing desired.

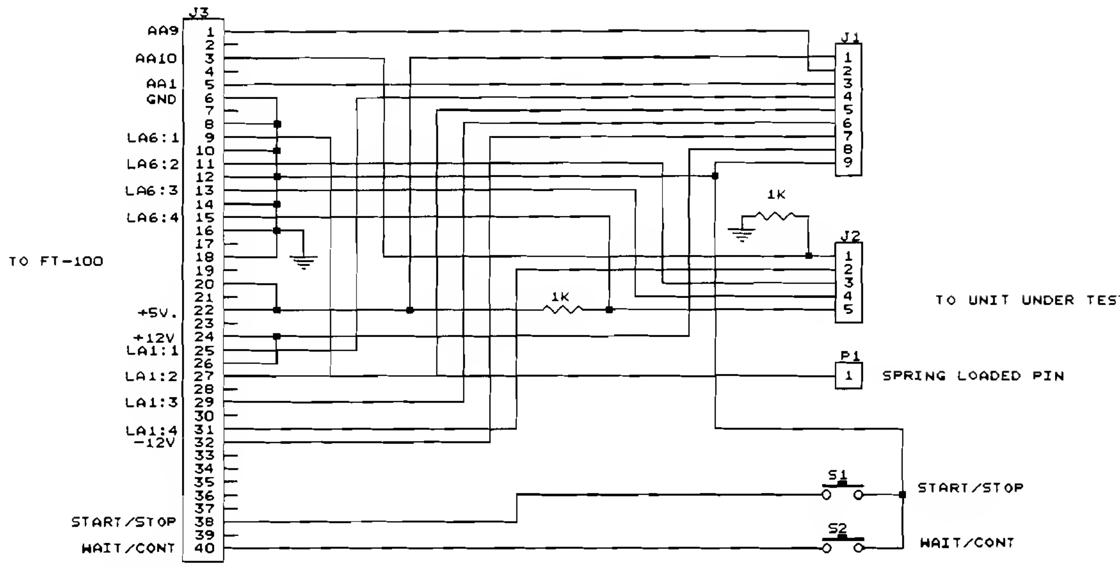
For this PC board, we will use a simple interface "box" that contains cables to plug into the board under test (since it has two connectors), and two pushbutton switches to allow control of the FT-100a test from the interface box. (START/STOP and WAIT/CONT).

The first step in setting up for a test is to define the desired input and output lines and how they are to be controlled or measured. Below is a diagram of the PC board connectors and the chosen I/O lines to connect to each. Note that there is a spring loaded test pin. This is to allow access to one point that does not go to either connector. The pin should be located such that it will contact the PC board on pin 3 of U3. A wire with a test clip or any other device that suits your particular needs could be used instead of a spring test pin. Also note that due to the somewhat limited I/O requirements, the 40 pin connector is all that is needed for this test.

<u>I/O INTERFACE DESCRIPTION</u>

Unit under test		FT-100a
J1 PIN 1	-	+5 VOLT SUPPLY
J1 PIN 2	-	AA9 (analog input line)
J1 PIN 3	-	AA1 (analog output line)
J1 PIN 4	-	LA1:1 (digital output line)
J1 PIN 5	-	LA1:2 "
J1 PIN 6	-	LA1:3 "
J1 PIN 7	-	-12 VOLT SUPPLY
J1 PIN 8	-	+12 VOLT SUPPLY
J1 PIN 9	-	GROUND (DC and analog)
J2 PIN 1	-	AA10 (analog input line)
J2 PIN 2	-	LA1:4 (digital output line)
J2 PIN 3	-	LA6:2 (digital input line)
J2 PIN 4	-	LA6:3 "
J2 PIN 5	-	LA6:4 "
Test pin	-	LA6:1

Schematic of Interface



Now that the hardware is ready, all that's needed is a program to run the test. Below is a listing of a program with comments and explanations. There are usually many different ways of achieving similar results. This program is not proposed as an ultimate solution, but merely an example of one way to test the PC board. This program assumes that a serial printer is connected to the serial port and a hardcopy of the test results is desired.

Program Listing

```
BD_TEST                 'Program Name
'
        CLS
'
'Initialize all I/O
'
        PA1=1
        AA1=0
        ER$="*** DEFECT *** "
'
'Print header information
        PRINT
        PRINT TAB 15; "************** FT-100a Board Test **************"
        PRINT " PC Board XYZ   Run Number ______________ "
        PRINT "Date: ";DATE$;TAB 20;"Time: ";TIME$
        PRINT
'
'See that all inputs are as expected
'
'Test analog inputs
        IF AA9 < 7.79 OR AA9 > 8.61 THEN CALL ER1
        IF AA8 < 4.5 OR AA8 > 5.5 THEN CALL ER2
'
'
'Test digital outputs
        IF LA6:1=1 THEN CALL ER3
        IF LA6:2=1 THEN CALL ER4
```

```
        IF LA6:3=1 THEN CALL ER5
        IF LA6:4=1 THEN CALL ER6
'
'Test relay contacts for open
        LA1:4=1: IF LA6:2=1 THEN CALL ER7
'
'Set variable resistor
        DISP "Turn R12 fully"                  'Instruct operator
        DISP "Counterclockwise"
        DISP "Press CONT"
        DISP "when ready."
        WAIT                                   'Set R12 & wait till done
'
        CLS: AA1=3.8                           'Set trip voltage
        DISP "Adjust R12 till"
        DISP "BEEP. If reach"
        DISP "end before beep,"
        DISP "press NO"                        'Use external YES-NO pod
        LA1:4=1
'
L2:     IF LA6:2=1 THEN BEEP: GOTO L1                  'Wait for R12 to be adjusted
        IF NOT NO THEN GOTO L2                         'Test NO pushbutton
'
        PRINT ER$; "R12 cannot be adjusted!"   'Defect in R12 adjustment
'
'Test LED circuit
L1:     CALL LED: IF RESULT=1 THEN CALL ER8
        LA1:1=0: CALL LED: IF RESULT=0 THEN CALL ER8
'
'Test inverters and flip-flop
        LA1:1=1                                'Set "D" at flip-flop to 1
        LA1:3=1                                'Release RESET line on flipflop
        LA1:2=1                                'Initiate latch procedure
        DELAY 15                               'Wait ≈15 ms. for C5 to discharge
        IF LA6:1=0 THEN CALL ER9               'See if inverters switched
        IF LA6:3=0 THEN CALL ER10              'See if flip-flop switched
'
'Test Completed
        CLS: DISP "Remove board": DISP "Attach Printout"
         DISP "Connect Next": DISP "Board to test.";
        BEEP 5000,250
'
        FOR VAR1 = 1 TO 10: PRINT: NEXT                'Scroll printer paper 10 lines
'
        END
'
'
'-------- LED on/off test subroutine -------
' Operator has direct interaction with program through the YES-NO pod.
' If LED is on, YES is pressed, 'otherwise, NO is pressed.
'
LED:    CLS: DISP "If LED is on,"
        DISP "press YES. If"
        DISP "off, press NO."
        RESULT=0
L3:     IF YES THEN INC RESULT: RETURN                 'YES?
        IF NOT NO THEN GOTO L3                         'NO?
        RETURN
'
```

```
'
'--- Error Subroutines ---
'
ER1:    PRINT ER$; "Voltage at pin 2 of J1 = ";AA9;".  Should be between 7.79 - 8.61"
        RET
'
ER2:    PRINT ER$; "Voltage at pin 1 of J2 = ";AA8;".  Should be above 4.5": RET
'
ER3:    PRINT ER$; "Pin 4 of U2B is high with pin 1 of U2A low.": RET
'
ER4:    PRINT ER$; "Pin 3 of J2 is high with open relay.": RET
'
ER5:    PRINT ER$; "Pin 4 of J2 is high with flipflop RESET.  (Pin 6 of J1 = 0)": RET
'
ER6:    PRINT ER$; "Pin 5 of J2 is high.  Should always be low (Ground line).": RET
'
ER7:    PRINT ER$; "Relay contacts are shorted.  High on pin 2 of J2 produces"
        PRINT TAB 15;"high on pin 3 of J3 (with relay open).": RET
'
ER8:    PRINT ER$; "LED is not switching properly.": RET
'
ER9:    PRINT ER$; "Inverters U2 A & B are not switching properly.": RET
'
ER10:   PRINT ER$; "Flip-flop U3 not latching properly.": RET
'
/
```

APPENDIX I EXAMPLE - DATA COLLECTION AND CONTROL CONFIGURATION

One of the more useful applications for the FT-100a is in Research and Development where control and monitoring of various circuits is needed. The FT-100a is well equipped to independently control and monitor a unit that needs to be tested for an extensive period of time. This is very difficult for a test or development engineer without help from some kind of control device.

The example presented here is a simple DC amplifier circuit that becomes unstable and occasionally oscillates at certain elevated temperatures. Because it is intermittent, it is very difficult for an engineer to see the problem, let alone take any test data during a failure.

The setup has the test amplifier located inside a temperature controlled environmental chamber. The input and output of the amplifier are connected to the FT-100a analog in and out lines. The engineer has determined that there are 5 voltages that need to be observed when the amplifier fails. These points are connected to other analog inputs in the FT-100a. The temperature need not be measured by the FT-100a since the environmental chamber program contains that data. The time of day will be necessary, however.

The following test is designed to run for 24 hours. If a failure is encountered, test data is taken and stored on a floppy disk.

CIRCUIT DESCRIPTION

The circuit is a very simple DC amplifier with a gain of 3.4 (+/- 0.2) and an input voltage range of 1.0 to 2.5 volts. The voltages being monitored are various points within the circuit and range from Ø to 9.0 volts. A more detailed description of the amplifier is unimportant for this test.

The I/O interface connection is very simple. The voltages to be monitored and the amplifier input are wired directly to the FT-100a analog input connector.

I/O INTERFACE DESCRIPTION

AMPLIFIER UNDER TEST	FT-100a
+9.0 volts	SUPPLY (Variable voltage supply)
Ground	GROUND (DC and analog)
Amp Input	AA1 (analog output)
Amp Output	AA9 (analog input)
Test volt# 1	AA10 "
Test volt# 2	AA11 "
Test volt# 3	AA12 "
Test volt# 4	AA13 "
Test volt# 5	AA14 "

Program Listing

```
AMP_TEST            'Program name
'
'Initialize before beginning test
'
        SUPPLY=9.0                  'Set the supply voltage
        OPEN "DATA1",OUT            'Open disk output file
        DEF_CNT=0                   'Use to count amount of defect data collected
'
        TSAVE$ = TIME$              'Save current time
        T_FLAG=0
'
'Start exercising the amplifier and testing for defect
'
BEG:    FOR VAR = 0 TO 20
        AA1 = (VAR * 0.075) + 1.0          'Scale analog output for 1 to 2.5 volts in 0.075v. steps
TEST:   IF AA9>=0.4*VAR OR AA9<=0.45*VAR THEN GOTO L1        'If amp passes test, go to L1
'
'------- Defect Routine ------
'
DEF:    STORE AA9,AA10,AA11,AA12,AA13,AA14      'Store all in/out data on disk
        STORE TIME$                             'Also store the time of day
        INC DEF_CNT                             'Add 1 to # of defects
        IF DEF_CNT>5000 THEN GOTO L2            'Quit taking data when get
                                                '5000 pieces of defect data.
        GOTO TEST                               'Continue taking data as long as defective
'
L1:     NEXT
        IF T_FLAG=1 AND TSAVE=TIME$ THEN GOTO L2      'Test for end of 24 hrs.
        IF TSAVE<>TIME$ THEN T_FLAG=1
        GOTO BEG
'
L2:     CLOSE                                   'Close disk file
        END
'
/
```

APPENDIX J CALIBRATION INFORMATION

To insure the accuracy of the analog input and output voltages over time, there are several adjustments within the FT-100a to compensate for minor component and environmental changes.

The FT-100a is completely calibrated at the factory and should not require recalibration for at least one year; however, the precise calibration schedule to follow must be determined by the needs and requirements of each individual application. In the absence of a more stringent calibration schedule, it is recommended that the FT-100a go through a complete calibration on a yearly schedule.

The FT-100a uses the same voltage reference source for the Analog Output circuitry (D/A) and the Analog Input circuitry (A/D). There are two analog voltage ranges, and each has an adjustment for its offset voltage. The procedures given here should be followed in the precise order to insure proper results.

The only equipment needed for a complete calibration is a high quality and accurate voltmeter (digital preferred). It is very important that the calibration of the voltmeter be correct since the FT-100 analog circuitry can be no more accurate than the voltmeter used for calibration. If a high-accuracy voltage source is available, the calibration results will likely be more precise, however satisfactory results may be obtained using the FT-100a as the voltage source.

> * * * * * CAUTION * * * * *
> There are dangerous and lethal voltages inside the FT-100! Do not touch any part of the power supply board or the AC power connector on the back panel!

The calibration of the FT-100a is broken down into steps. It is not necessary that these calibrations be performed in any given order. However, if the FT-100a is to be used as a voltage source for calibration of the analog voltage inputs, the reference voltages and analog output gain calibrations must accurate.

1. Reference Voltages - There are two ranges, each with reference voltage adjustments.
2. Analog Output Gain - Each of the seven analog outputs have a gain adjustment.
3. Programmable Power Supply - This is simply a voltage adjustment.
4. Analog Input Gain - All the analog inputs are multiplexed into one amplifier circuit. The gain of this amplifier must by set.

Before calibration can begin, the FT-100a cover must be removed. Unplug the FT-100a unit and remove the cover (8 screws). After the cover is removed, reconnect AC power and turn on the unit. **Be very careful. There are dangerous voltages inside the unit!**

Calibrating the Reference Voltages

1. Connect the negative voltmeter lead to one of the ground lines in the 40 pin I/O Interface (front panel, center connector).
2. Connect the positive voltmeter lead to the AA1 analog output line (pin 5 of the 40 pin connector).
3. The output voltage at AA1 will need to be set to several voltages. This may be done through Direct Commands, or the following program may be loaded and run to simplify the process:

```
REF_SET                 'Program name
'
BEGIN: V_HI=10
L1:     ANALOG V_HI : AA1 = 0                      'Set voltage range & set output = 0
L0:     CLS : DISP "V Out=";AA1                    'Display voltage out
        WAIT                                       'Wait for operator
        IF YES THEN GOTO BEGIN                     'If YES pressed, select 0-10v range
        IF NO THEN V_HI = 5 : GOTO L1              'If NO pressed, select +/-5v range
        IF AA1 < V_HI THEN AA1 = AA1 + 1: GOTO L0  'Increase voltage out by 1 volt
        AA1 = V_HI - 10 : GOTO L1                  'Set voltage to lowest value
/
```

Note:
This program will set analog output AA1 and display the voltage on the LCD. Each time the WAIT pushbutton is pressed, the output voltage at AA1 is increased by 1.0 volt. When the output voltage reaches the upper limit of its range, the output is reset back to the lower limit. If the YES pushbutton is pressed while WAIT is pressed, the voltage range will be 0-10 volts. If the NO pushbutton is pressed while WAIT is pressed, the voltage range will be +/-5 volts. This program provides a simple way of changing the output voltage, but the same job may be done through Direct Commands.

4. Set the voltage range to 0-10 volts, and set the analog output voltage to 2.0 volts (either using the above program or through a Direct Command).
5. Using a small tuning tool, adjust the trimmer pot R2 until the voltage output is equal to 2.0 volts.
6. Now set the analog voltage range to +/-5 volts and AA1 to 1.0 volts.
7. Adjust trimmer pot R3 until the output voltage is equal to 1.0 volts.
8. This completes the Analog Offset Voltage adjustments.

Analog Output Gain Calibration - Reference Voltages <u>must</u> be calibrated prior to gain adjustments. This calibration requires the setting of several analog outputs. Direct Commands may be used for this task, or a simple program could be developed.

1. Set the analog output range to 0-10v. (ANALOG 10)
2. Connect negative lead of voltmeter to one of the analog ground lines in the 40-pin I/O connector, and connect the positive voltmeter lead to AA1 output in the 40-pin I/O connector.
3. Set analog output AA1 to 10 volts.
4. Adjust trimmer pot R13 for the voltmeter to "flicker" between 10 volts and the next lower voltage.
5. Continue setting each analog output to 10.0 volts and adjust each gain control, monitoring the outputs.
   Note: Analog outputs AA1-AA4 are 12-bit resolution, while outputs AA5-AA7 and the programmable supply are 8-bit resolution. Be aware of the resolution when attempting to adjust the gain.

Use the following chart for the analog output gain adjustments:

Set this Analog Output to 10.0 volts	Adjust this trimmer pot
AA1	R13
AA2	R14
AA3	R15
AA4	R16
AA5	R20
AA6	R21
AA7	R22

8. The analog output gain adjustment is complete. It is usually a good idea to test the outputs at several different voltages to confirm the calibration over the complete output range. Make any additional adjustments as needed.

Programmable Power Supply Calibration

1. Connect the voltmeter positive lead to the programmable supply output at the 40-pin I/O connector.
2. Set the SUPPLY output to 10.0 volts, and adjust trimmer pot R46 for the voltmeter to "flicker" between 10 volts and the next lower voltage.
3. This completes the Programmable Power Supply calibration.

Analog Input Gain Calibration - The only required adjustment is the input amplifier gain control. Be sure the voltage references are calibrated prior to beginning this calibration. A good voltage source is required for the input gain calibration. One of the FT-100a analog outputs may be used for this source if no other source is available.

1. Connect the voltage source to AA9 input (in the 40-pin I/O connector).
2. Set the Analog Voltage Range to 0-10 volts (ANALOG 10).
3. Set the voltage source to 10.0 volts.
4. Read the voltage at input AA9 using either Direct Commands or a simple program. The program below can be used to read this input:

```
VOLT_RD               'Program Name
'
BEG:   CLS 2 : DISP AA9 : GOTO BEG
/
```

5. Adjust trimmer pot R43 for the reading to "flicker" between 10.0 and the next lower reading.
6. This completes the Analog Input Gain adjustment. It would be a good idea to verify the calibration for all input channels, and at several voltages. Make readjustments as necessary.

APPENDIX K DIFFERENCES BETWEEN FT-100 and FT-100a

The FT-100a is an upgrade and improved version of the original FT-100. All programs, I/O devices, and Expansion Modules that interfaced to the FT-100 will also perform flawlessly with the FT-100a. The reverse of this statement is not necessarily true. If a program has been developed for the FT-100a, use caution if this program is to be run on an FT-100.

Below are some of the most notable improvements that can be found in the FT-100a:

Keyboard Input - The FT-100a has a direct keyboard input. This facilitates the entry of data into a test program. It also permits Direct Command Mode operation without an additional computer or terminal.

Multiple Analog I/O Range - The programmer can set the voltage range for the analog I/O to either 0-10 volts or +/-5 volts. The range can even be changed during program execution.

12-bit Analog I/O - The Analog Inputs are all 12-bit resolution. There are four Analog Outputs with 12-bit resolution, and three with 8-bit resolution.

High-impedance Analog Inputs - All the Analog Input lines are ultra-high impedance. There is virtually no loading on even the most sensitive circuits being measured.

Higher Baud Rate - The FT-100a is capable of reliable serial communications at 19.2K baud rate.

Index

F

G

H

I

K

L

M

N

O

P

App. Note#: AN-0100-01 Date Issued: November, 1994

Subject: Serial Communications Product: FT-100, FT-100a

The FT-100's serial communications port is very versatile and may be used for many different functions, and in many different ways. This application note's purpose is to describe the FT-100 serial port in more detail and go into more depth about how to use if for the varied functions.

There are several facts concerning the serial port that might be helpful:

- Communications is always half-duplex. In other words, data can go only one direction at any given time. If data is being sent, all incoming data is ignored.

- RTS/CTS handshaking is always used for data transmitted out the serial port. There is no handshaking for incoming data.

- The serial input line is connected to a hardware interrupt line in the FT-100. If interrupts are disabled, data cannot be received. The only exception to this is when the Direct Command Mode is enabled and a GET command is used to input one character.

- The serial input buffer can hold up to 128 characters. If an attempt is made to input more than 128 characters, the BUFFER OVERFLOW error will result. In most cases, when data is read from the serial port, the buffer is cleared. The exception to this occurs when the GET command is used with Direct Command Mode disabled. Only one character is removed from the buffer in this case.

- Any time Direct Command Mode is enabled, and a carriage return (ASCII ØDH) is received, the input character string will be interpreted as a direct command. Also, with Direct Command Mode and Echo enabled, the backspace character (Ø8) will cause the FT-100 to send the following character string: Ø8, 2ØH, Ø8 (backspace, space, backspace).

- The serial input buffer is cleared of all data in the following conditions:
    - Program Starts
    - Program Ends
    - Program is Stopped
    - Error condition
    - Direct Command Mode status is changed (turned on, or turned off)
    - Beginning of the IN command
    - Beginning of the GET command if the Direct Command Mode is enabled
    - Whenever the serial port is switched. Note: This is still true even if a 1041 Serial Port Expander is not connected.

- If the Direct Command Mode is disabled, and there has been incoming data, and a WAIT command is executed (either WAIT pushbutton or program command), the data will remain in the serial input buffer. If an attempt is made to execute a direct command while in the "wait" state, there will most likely be a SYNTAX error, since the data in the buffer will be interpreted as a command.

SERIAL COMMUNICATIONS COMMANDS

Following are the commands that interact with the serial port, and a brief description of each:
(NOTE: See the operators manual for detailed information on these commands)

PRINT - Sends character strings out the serial port. While characters are being transmitted, all incoming data is ignored until the transmission is finished. If the FT-100 sensed incoming data at the end of the transmission, it will immediately begin receiving. This can cause erroneous results if the FT-100 begins receiving in the middle of a character.

IN - Clears the serial input buffer and waits for a character string ending with a carriage return (ASCII ØDH). Interrupts are not disabled during the IN command, but the Direct Command Mode is disabled.

GET - Gets one character from the serial port. The GET command functions differently, depending on whether or not the Direct Command Mode is enabled.

Direct Command Mode Enabled

When the GET command begins, the input buffer is cleared. The program then waits for one character to be input. This character is placed in the referenced string variable. If no character is received, the only way to get the program out of the GET command is to press the RESET pushbutton. For this reason, it is usually advisable to disable the Direct Command Mode when using the GET command to input characters.

Direct Command Mode Disabled

All serial data is placed in the serial input buffer in a first-in, first-out fashion. The buffer can hold up to 128 characters at any given time. The GET command will "get" the first character and place it in the referenced string variable. Interrupts must be enabled for data to be received into the buffer.

If the Direct Command Mode is disabled, and a WAIT command is issued (WAIT pushbutton or program command), there will likely be characters in the input buffer. If attempt is made to issue a direct command, a SYNTAX error will usually result.

TWO-WAY COMMUNICATIONS

One of the safest methods used when two devices must communicate with each other is to use a form of software handshaking. This can usually be as simple as placing a never-used character on the end of the data string to signal to the receiving device that it may begin sending data. Two FT-100's can easily communicate with each other using this method.

Most serial communications troubles are caused by timing problems. This is especially true when using a half-duplex system with no handshaking. Be sure to allow enough time for the receiving end to process its data. Also, be aware of the contents of the input buffer at all times. This also includes knowing when the buffer is cleared, and when it is not.

Good luck.

-tek, Inc. --- APPLICATION NOTE ---

App. Note#: AN-0100-03 Date Issued: February, 1995

Subject: ANALOG BAR GRAPH Product: FT-100, FT-100a

Sometimes it is very convenient to be able to "see" a voltage change by using a bar graph. This simulates an analog meter and is often helpful when setting a voltage and especially when a voltage must be tuned for either maximum or minimum value. The following is a very simple routine that will display a voltmeter bar graph on the FT-100 LCD display. The first part of the routine is for setup purposes and should be placed near the beginning of your main program (before you call the subroutine METER). Any time you wish to use the bar graph, simply set up the input parameters for the subroutine and use the command CALL METER. The subroutine is set up to return <u>only</u> when the NO pushbutton is pressed.

The routine is very simple, and you should have no trouble customizing it for your particular applications. Be very careful to copy all commands and data statements exactly. The comments are only to clarify the routine and may be left out.

```
METER2          'Program name
'
        PROG_LEN=100            'Number of bytes in assembly program
'
'------ See if there is enough memory to load in assembly program ------
        LAB_BEG=PEEK 45+(PEEK 46*256)+((PEEK 47+(PEEK 48*256))*16)
        STR_BEG=PEEK 59+(PEEK 60*256)+((PEEK 61+(PEEK 62*256))*16)
        IF LAB_BEG<(STR_BEG+PROG_LEN)+16 THEN BEEP:CLS:DISP:DISP "NOT ENOUGH":DISP " MEMORY!":END
'
        PROG_SEG=INT (STR_BEG/16)+1:PROG_OFF=0:ADDR=PROG_SEG*16   'Address for assemby routine
'
        FOR I=0 TO PROG_LEN:READ X:POKE (I+ADDR),X:NEXT           'Load in the assembly routine
'
        SYS PROG_SEG,PROG_OFF                   'Call the assembly routine (set up LCD characters)
'
'---- These Data statements are the assembly routine ----
        DATA 199,6,108,1,118,4,187,45
        DATA 0,178,72,182,7,185,8,0
        DATA 154,104,10,0,192,138,194,230
        DATA 7,254,194,154,104,10,0,192
        DATA 46,138,7,230,135,67,226,232
        DATA 254,206,117,225,203
        DATA 16,16,16,16,16,16,16,0
        DATA 24,24,24,24,24,24,24,0
        DATA 28,28,28,28,28,28,28,0
        DATA 30,30,30,30,30,30,30,0
        DATA 31,31,31,31,31,31,31,0
        DATA 16,16,16,31,0,0,0,0
        DATA 1,1,1,31,0,0,0,0
'------------------------------------------------
'
```

```
'
'          This section sets up the parameters for calling the METER subroutine.  These
'          parameters may be set anywhere in your program as long as they are set prior
'          to calling METER.  Be sure parameters are within their limits, also V_MIN must be
'          less than V_MAX.
'
           LINE=9              'LINE is the analog input line in module A (9-16)
           V_MIN=0             'V_MIN   is the low end of the voltage scale to be displayed (0-10)
           V_MAX=10            'V_MAX   is the high end of the voltage scale to be displayed (0-10)
'
           CALL METER          'Call bar graph display subroutine. This command can be anywhere.
'
           END
'
'
'---------- Bar graph voltmeter display subroutine -----------
'
'          This subroutine should be place in a location where it will not interfere with your
'          main program, such as at the end of the main program.
'
METER:  CLS:CURSOR 2,1:DISP CHR$ 6;"------+------";CHR$ 7           ' This section sets up
        DISP RIGHT$ (STR$ V_MIN,(LEN STR$ V_MIN-1));                ' the voltmeter scale
        V$=RIGHT$ (STR$ V_MAX,LEN STR$ V_MAX-1)                     ' on the LCD display.
        DISP TAB (16-LEN V$);V$                                     ' It could be changed
        INCR_S=(V_MAX-V_MIN)/75:INCR_L=INCR_S*5                     ' to suit your needs.
        CUR=1:PTR=1
'
L6:     V_OLD=V
L5:     IF NO THEN RETURN                 'If NO is pressed, get out of subroutine
        V=AA(LINE)                        'Read analog input line (Note: Change for different module)
        IF V=V_OLD THEN GOTO L5           'If voltage reading is same as last reading, do nothing
'
        CURSOR 4,1:DISP V;TAB 11;"volts";          'Display voltage (Character display)
'
'--- See if voltage is within scale of meter
        IF V<V_MIN THEN CLS 1:DISP CHR$ 127;:GOTO L6
        IF V>V_MAX THEN CURSOR 1,1:FOR I=1 TO 15:DISP CHR$ 5;:NEXT:DISP CHR$ 126;:GOTO L6
'
'--- Display bar graph of voltage
        VV=V-V_MIN:CURSOR 1,1:CUR=1
L7:     IF VV>INCR_L THEN DISP CHR$ 5;:VV=VV-INCR_L:INC CUR:GOTO L7
        DISP CHR$ (INT (VV/INCR_S));
        FOR I=CUR TO 15:DISP " ";:NEXT
        GOTO L6
/
```

NOTICE: Every effort has been made to insure the accuracy of the information contained in this document, however Y-tek is not responsible for any consequences resulting form erroneous or inaccurate information.

Y-tek, Inc. --- *APPLICATION NOTE* ---

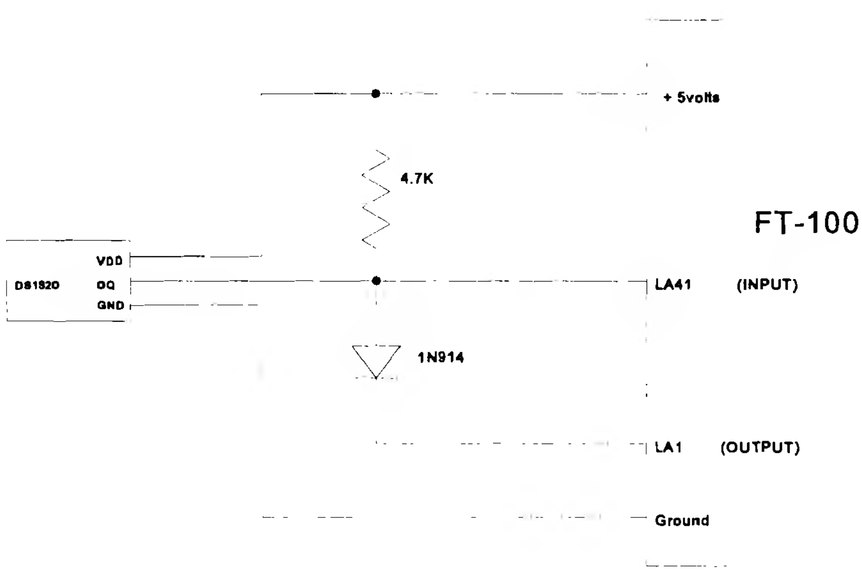
App. Note#: AN-0100-04 Product: FT-100 Date Issued: February, 1995

Subject: TEMPERATURE MEASUREMENT USING DALLAS DS1820

This application note describes a method of interfacing the FT-100 to the Dallas Semiconductor DS1820 Digital Thermometer. This is one of the simplest and least expensive methods of temperature measurement using the FT-100, plus the DS1820 is pre-calibrated and accurate to +/-0.5°C and operates from -55°C to +125°C. Temperature conversion takes about one second, typically.

Only two of the digital I/O lines are used, and the FT-100 supplies all power for the DS1820. The DS1820 cost around $5.00 and the necessary circuitry consists of only one resistor and one diode. Because the output from the DS1820 is digital serial data, and not a voltage or current, the interface cable can be relatively long and not overly susceptible to voltage drop or noise. Twisted wire cable should provide sufficient noise immunity. If there is any doubt, experiment extensively with different cable lengths and types. The location of the resistor and diode in the interface is not critical. If you do not have access to a DS1820 or data sheets, contact Y-tek.

This subroutine uses digital lines LA1 (output) and LA41 (input) for the interface. These I/O lines cannot be changed very easily since they are specified in the assembly routine. If you must use other lines, contact the factory for help.



SCHEMATIC OF DS1820 INTERFACE

Temperature Read Software

This routine is very simple, and you should have no trouble customizing it for your particular applications. Be very careful to copy all commands and data statements exactly. The comments are only to clarify the routine and may be omitted.

```
TEMP_RD
'
        PROG_LEN=124            'Number of bytes in assembly program
'
'--- Test for enough memory to load in assembly routine ---
        LAB_BEG=PEEK 45+(PEEK 46*256)+((PEEK 47+(PEEK 48*256))*16)
        STR_BEG=PEEK 59+(PEEK 60*256)+((PEEK 61+(PEEK 62*256))*16)
        IF LAB_BEG<(STR_BEG+PROG_LEN)+16 THEN BEEP:CLS:DISP:DISP "NOT ENOUGH":DISP "  MEMORY!":END
'
        PROG_SEG=INT (STR_BEG/16)+1:PROG_OFF=0:ADDR=PROG_SEG*16   'Address for assembly routine
'
        FOR I=0 TO PROG_LEN:READ X:POKE (I+ADDR),X:NEXT            'Load in assembly routine
'
'------------------------------ Assembly Program ------------------------------
        DATA 0E8H,4AH,00,0B2H,0CCH,0E8H,57H,00,0B2H,44H,0E8H,52H,00,0E4H,00,0D0H
        DATA 0E8H,73H,0FAH,0E8H,37H,00H,0B2H,0CCH,0E8H,44H,00H,0B2H,0BEH,0E8H,3FH,00
        DATA 0B4H,10H,0E8H,20H,00H,0CH,01,0E6H,00,2BH,0C9H,2BH,0C9H,0E4H,00,0B9H
        DATA 28H,00,0E2H,0FEH,0D0H,0E8H,0D1H,0DAH,0FEH,0CCH,75H,0E6H,0E8H,0EH,00,0BBH
        DATA 7EH,01,89H,17H,0CBH,0A0H,7BH,00,24H,0FEH,0E6H,00,0C3H,0E8H,0F5H,0FFH
        DATA 0B9H,27H,01,0E2H,0FEH,0CH,01,0E6H,00,0B9H,27H,01,0E2H,0FEH,0C3H,0B4H
        DATA 08,0E8H,0E1H,0FFH,8AH,0F2H,80H,0E6H,01,0AH,0C6H,0E6H,00,0B9H,32H,00
        DATA 0E2H,0FEH,0CH,01,0E6H,00,0D0H,0EAH,0FEH,0CCH,75H,0E5H,0C3H
'------------------------------------------------------------------------------
'
' This is a simple program to display the temperature on the LCD display and
' is only for demonstration purposes. You would probably want to call the
' subroutine and use the temperature reading in other ways.
'
        CLS:DISP "Temp:      ";CHR$ 0DFH;"C":DISP TAB 14;CHR$ 0DFH;"F"     'Set up LCD display
'
L0:     CALL READ_T                                  'Call subroutine to get a temperature reading
        IF TEMP=OLD_T THEN GOTO L0                   'If temperature has not changed, keep reading
        CURSOR 1,7:DISP "     ":CURSOR 1,7:DISP TEMP 'Display degrees C.
        DISP "          ":CURSOR 2,7:DISP TEMP*1.8+32 'Display degrees F.
        OLD_T=TEMP:GOTO L0                           'Save temperature reading to compare with next reading
'
'
'----------- Subroutine to read temperature -----------
'       Subroutine will take one reading of the temperature and return the
'       value in the variable TEMP. TEMP is in degrees C.
'
READ_T: SYS PROG_SEG,PROG_OFF                            'Read Remote Temperature Sensor
        IF (PEEK 17FH)=0 THEN TEMP=(PEEK 17EH)/2:RETURN  'Positive Temperature?
        TEMP=-(256-(PEEK 17EH)/2)                        'Negative Temperature
        RETURN
/
```

NOTICE: Every effort has been made to insure the accuracy of the information contained in this document, however Y-tek is not responsible for any consequences resulting from erroneous or inaccurate information.

App. Note#: AN-0100-05 Product: FT-100 Date Issued: April, 1995

Subject: TESTING LOW-CURRENT CMOS INTEGRATED CIRCUITS

Quite often there is a need for a simple and inexpensive method of pre-screening CMOS logic integrated circuits before they are installed in a circuit board assembly. Pre-testing these IC's is especially beneficial when they are to be installed into surface mount assemblies or multi-chip modules (MCM). Troubleshooting can be very difficult and replacement is often harmful to the board assembly.

This application note describes a method of testing CMOS logic gates that is very simple and easy to implement. Because the requirements for each particular application will vary, this example uses a 74HCT86 CMOS XOR gate package for illustration only. Other types of logic IC's could be tested using similar methods. The test parameters would be different, and input/output lines may change, but the methodology would remain pretty much the same. By studying this example, and seeing how the FT-100 is used to control the device under test and measure the low currents, it should be relatively easy to derive your own tests.

Because this test needs more analog input lines than the FT-100 main unit can provide, we are using a 1504 Expansion Module, which gives us 16 extra analog inputs with an input range of 0-5 volts. Other expansion modules could be used as well, but the 1504 is the least expensive Expansion Module that can accomplish our task.

Test Requirements

The test parameters for our IC are the following:

- With no load on the outputs and all inputs set to zero volts, the overall device current must be less than 10 µa.
- Current for each gate input must be less than 1 µa.
- Each gate output must be able to sink and source 4 ma.
- Gates must have the proper logic output.
- Test must stop on the *first* defect. No need to test remainder of device if there is a defect.

These parameters will obviously change with each type of device and especially with a different family type. Once again, the 74HCT86, and these parameters, are used only as an example.

Circuit Description

Refer to the overall schematic in figure 1. In order to measure current, we must convert the current into a voltage by feeding it through a resistor of known value. We can then measure the voltage drop across the resistor and mathematically determine the current (remember Ohms law?). The resistors should be high precision (1% or better), and depending on individual tolerance requirements, they may need to be adjustable for very fine calibration. Since the overall current to the device must be less than 10µa, we can insert a 100KΩ resistor (R17) in series with the VCC line and measure the voltage drop across the resistor. If the voltage is greater than 1.0 volts, we know the device current is over 10 µa. To insure that VCC does not drop or change during the other tests, we use digital output line LA9 and a diode to provide a higher current VCC.

In a similar fashion we test the gate input currents by inserting a 1 MegΩ resistor in series with each input. By measuring the voltage drop across the input resistors, we can derive the input current. This current must be less than 1 μa, so the voltage across the 1 MegΩ resistor must be less than 1.0 volts. The gate inputs can be driven high or low with the input current tested in both configurations.

The gate outputs must be able to sink and source at least 4 ma. To test this, we place a resistor load across each gate output and test the output voltage. This allows us to "connect" pull-up or pull-down resistors of know value and determine the output current drive capabilities.

Hopefully, this test example will be sufficient to show how the FT-100 can be used to measure very small currents when necessary. Inserting series resistors in the gate inputs will obviously "slow" the overall gate reaction. If this is a problem, a small delay may be necessary in the test program after each input setup to give the gate input voltage time to reach its ultimate level. If the gate <u>must</u> operate at full speed, it would be very easy to use analog switches, such as the DG202 to effectively bypass the series resistance. If this is a concern, contact Y-tek for applications assistance.

If you have components to test and aren't sure how to go about setting it up or how the FT-100 can be used, give us a call at Y-tek. An Applications Engineer will be glad to help you.

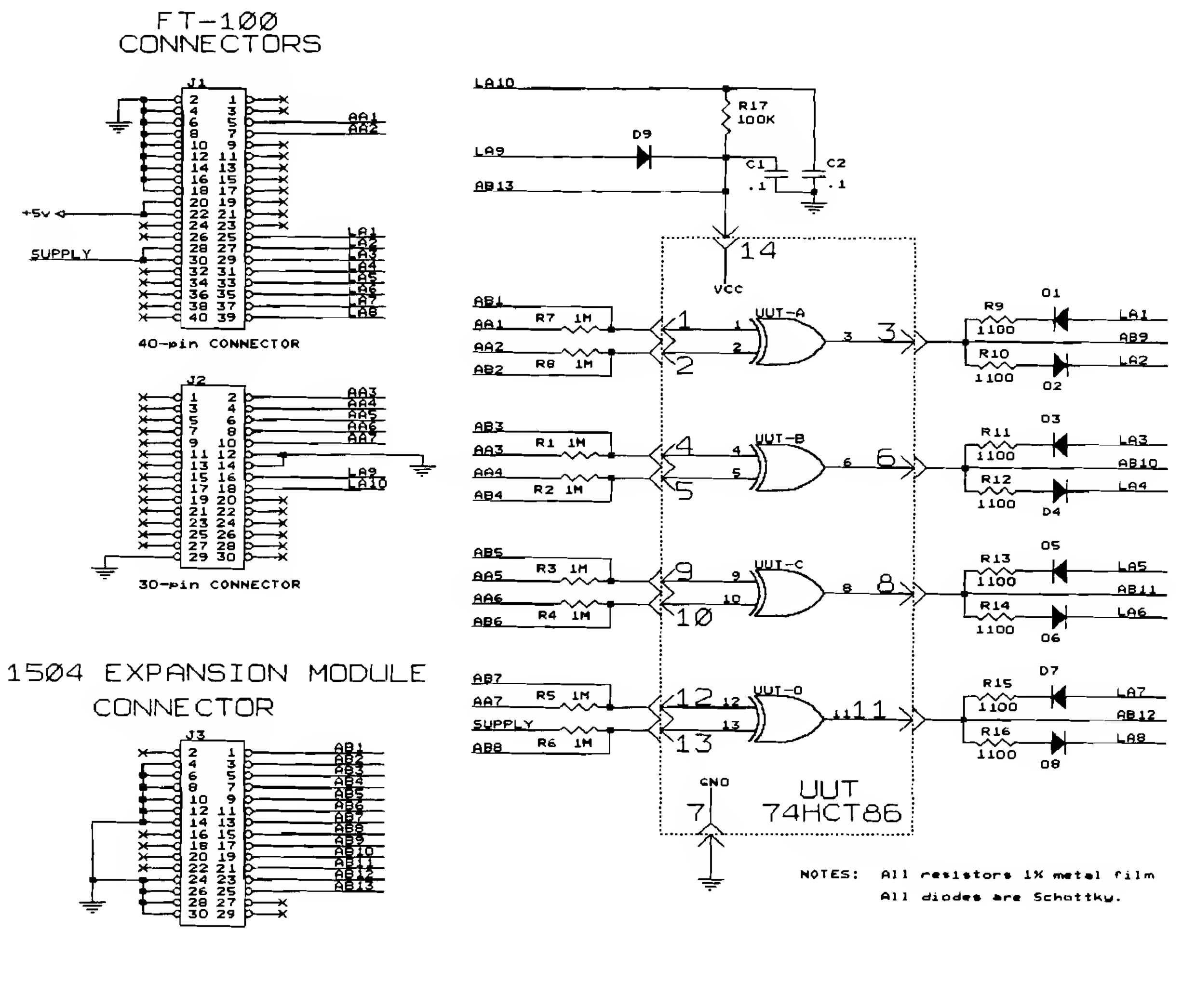


FIGURE 1 - Schematic of CMOS digital gate test fixture

FIGURE 2 - Program Listing

```
GATE_TST                'Program Name
'
' This test program will functionally test the designated CMOS gate for proper operation, and also
' measure the following:
'           - Current drain on each input line.
'           - Overall current drain on the VCC line.
'           - Output sink and source voltage from each gate with and without a designated load. I/O line.
'
' All test results will be printed on a serial printer (or sent to a data terminal or computer).  The LCD display
' will be used only for operator instruction.
'
'
        D$=">>> DEFECT! <<< "
'
        FOR X=0 TO 3:READ Q(X):NEXT          'Read in the correct logic output for gate under test
        DATA 0,1,1,0                         'Data for gate under test
'
BEG:    PRINT:PRINT:PRINT "* * * * * * * * * * XOR GATE IC TEST * * * * * * * * * *"        'Print header
'
'---------- Initialize all I/O lines to a known state ------------
        LA9=0                                'LA9 acts as a high current VCC source.
        LA10=0                               'LA10 acts as the low current VCC source.
        PA1=10101010B                        'This removes all sink and source load from the gate output.
        FOR X=1 TO 8:AA(X)=0:NEXT            'Set analog output lines to 0.0 volts.
'
'------ Instruct operator that test is ready to run ------
        CLS: DISP "Insert IC, then": DISP "Press YES": DISP "Press NO to":DISP "end test."
L0:     IF NO THEN CLS:END                   'End of test
        IF YES=0 THEN GOTO L0                'Wait for YES to be pressed.
'
'
'--- TEST #1: Measure the current drawn by VCC line & test for less than 10 micro-amps ---
        LA10=1                               'Turn on low current VCC
        IF AB13>4 THEN GOTO L1               'Test for more than 1 volt drop across R17
          LA10=0                             'Turn off low current VCC
          PRINT D$;"Overcurrent on VCC line!  VCC current = ";(5-AA13)/0.1;" micro-amps"
          GOTO DEFECT                        'Go to defect handling routine.
'
'
'--- TEST #2: Test gates for proper logic output ---
        LA9=1                                'Turn on high current VCC
L1:     FOR GATE=1 TO 8                                         'Gate counter
         FOR X=0 TO 3
           AA(GATE)=(X AND 1B)*5: AA(GATE+1)=(X AND 10B)*5      'Set two gate inputs
           IF Q(X)=INT (AB(GATE+8)/4.0) THEN GOTO L2
               PRINT D$;"Gate #";GATE-INT(GATE/2);              'Gate defect
               PRINT ":    Inputs =";X AND 1;" ";X AND 10B;"  Output =";AA(GATE+8);"  Should =";Q(X)
               X=3:GATE=8:NEXT:NEXT:GOTO DEFECT                 'Abort test
L2:      NEXT
        INC GATE:NEXT                   'Done with all gates?
        FOR X=1 TO 8:AA(X)=0:NEXT            'Set all gate inputs back to zero
'
```

```
'
'--- TEST #3:  Measure & Test input current at gate inputs for less than 1 micro-amp ---
' (First test source input current)
        FOR LINE=1 TO 8                                               'Test all 8 inputs
        AA(LINE)=5:IF (AA(LINE)-AB(LINE))<1.0 THEN GOTO L3      'Voltage across resistor < 1.0 v.?
                PRINT D$;"Gate #";LINE-INT(LINE/2);
                PRINT "  Over-current!  Gate input drive = high."
                PRINT "    Current at AA";LINE;" is: ";AA(LINE)-AB(LINE);" micro-amps."
                LINE=8:NEXT:GOTO DEFECT                                       'Abort test
L3:     NEXT
' (Now test sink input current)
        FOR LINE=1 TO 8                                               'Test all 8 inputs
        AA(LINE)=0:IF (AB(LINE)-AA(LINE))<1.0 THEN GOTO L4      'Voltage across resistor < 1.0 v.?
                PRINT D$;"Gate #";LINE-INT(LINE/2);
                PRINT "Over-current!  Gate input drive = low."
                PRINT "    Current at AA;"LINE;" is: ";AB(LINE)-AA(LINE);" micro-amps."
                LINE=8:NEXT:GOTO DEFECT                                       'Abort test
L4:     NEXT
'
'
'--- TEST #4:  Test gate outputs for proper current drive capabilities ---
'
' (First turn on pull-up resistors & test for sink current)
        PA1=0FFH                                'Turn on pull-up loads on gate outputs
        FOR GATE=1 TO 4                         'Test all 4 gates
        IF AB(GATE+8)<=0.4 THEN GOTO L5         'Test output voltage
                PRINT D$;"Gate #";GATE;"  With pull-up resistor, output should be 'low'."
                PRINT "   Output =";AB(GATE+8);" volts."
                GATE=8:NEXT:GOTO DEFECT                                       'Abort test
L5:     NEXT
'
' (Now turn on pull-down resistors & test for source current)
        PA1=0                                   'Turn on pull-down loads on gate outputs
        FOR X=1 TO 8:AA(X)=5:INC X:NEXT         'Set gate inputs to yield 1 on output
        FOR GATE=1 TO 4                         'Test all 4 gates
        IF AB(GATE+8)>=4.1 THEN GOTO L6         'Test output voltage
                PRINT D$;"Gate #";GATE;"  With pull-down resistor, output should be 'high'."
                PRINT "  Output =";AB(GATE+8);" volts."
                GATE=4:NEXT:GOTO DEFECT                                       'Abort test
L6:     NEXT
'
        PRINT "<<<< PASSED >>>>"
        GOTO EOT                'End of test - <<PASSED>>
'
'
'
'---------- Defect Routine ----------
DEFECT:  FOR X=1 TO 4:BEEP 5000,50:DELAY 20:NEXT             'Alarm signal
EOT:     PRINT "---------------------- END OF TEST ----------------------"
         GOTO BEG
'
/'
```

NOTICE: Every effort has been made to insure the accuracy of the information contained in this document, however Y-tek is not responsible for any consequences resulting form erroneous or inaccurate information.

-tek, Inc. --- APPLICATION NOTE ---

App. Note#: AN-0100-06 Product: FT-100 Date Issued: April, 1995

Subject: SCALING DC INPUT VOLTAGES

Whenever a DC voltage is converted into digital format, there is always some loss of resolution due to the incremental nature of the conversion. For instance, an 8-bit converter will divide the input voltage range into 256 increments, or voltage steps. If the input voltage range is 0 to 10.0 volts, this will yield increments of about 39.1 mv. A 12-bit converter will improve this resolution considerably by dividing the input scale into 4096 increments. For the same 0 to 10.0 volt input, the increment would be 2.44 mv. It is easy to see that increasing the "bit resolution" of the analog converters will increase the resolution.

This application note describes some "tricks" that are very easy to implement and can increase the effective bit resolution of your input voltage scale. For many DC voltage measurements, you do not need the complete voltage range that the FT-100 provides. If the voltage range of interest is scaled to the full FT-100 input voltage range, the resolution of the reading will increase. Suppose, for instance, that your circuit has a reference voltage that must be set to 1.20 +/-0.01 volts. Using the 0-10 volt input on the FT-100 only allows you to reach the target within +/-39.1 mv. By scaling an input range of 0.5-1.5 volts to 0-10.0 volts, we have effectively decreased the incremental steps to 3.91 mv. - using only an 8-bit analog converter!

Also, suppose you want to measure a voltage that is outside the range of the FT-100 (or Expansion Modules). The circuits shown here will allow you to scale almost any voltage range. After scaling, the test program will need to compensate for the scaled readings. The compensation formulas are given here as well. Using these circuits allow you to take full advantage of the analog input measurement capabilities of the FT-100.

The circuits shown here are very universal and should cover most voltage scaling applications. In general, the operational amplifiers shown may be any universal op-amp. Depending on the accuracy desired, you might need to consider the individual op-amp characteristics, especially such things as the input offset voltage, input voltage range, etc.. Because all components have tolerances, trimmer potentiometers would be recommended for the gain feedback resistor (R2 in figure 1). Depending on the overall accuracy desired, the bias voltage may need to be derived from a precision voltage source.

There are a few things that should be considered when designing the scaler circuit:

- The input impedance is determined almost solely by R1. Try to make R1 as large as practical. Values up to 100KΩ should work fine. If a higher input impedance is desired, use another op-amp as a voltage follower.
- The bias voltage (V_{BIAS}) must be within the working voltage limits of the op-amp input. (V_{BIAS} between -10.5 and +10.5 volts should work fine).
- The bias voltage must be derived from a voltage source as stable and accurate as your measurements dictate. The voltage reading will be only as accurate as your bias voltage.
- Resistors should be low tolerance, high quality - 1% metal film.
- By-pass capacitors should be used liberally.
- Ground lines should be hefty and as short as possible.

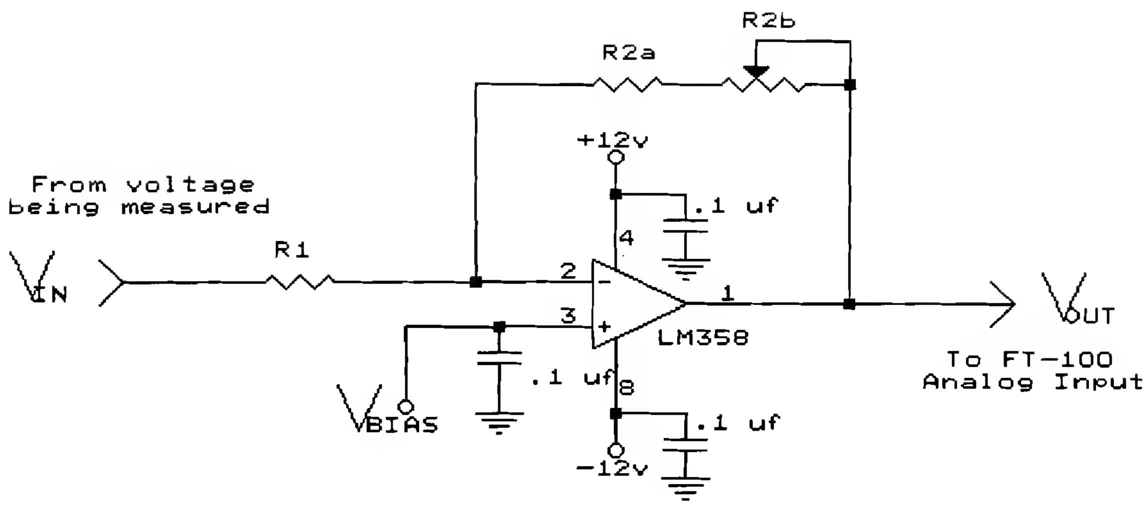


FIGURE 1 - DC Voltage Scaler

The first step to designing your scaling circuit is to determine the overall voltage gain (A_V). The voltage gain is defined as:

$$A_V = ABS\left(\frac{\text{Output voltage range}}{\text{Input voltage range}}\right) = ABS\left(\frac{(V_{OH} - V_{OL})}{(V_{IH} - V_{IL})}\right) \qquad \text{(EQUATION 1)}$$

Where: A_V = DC voltage gain.
V_{IH} = High end of input voltage range.
V_{IL} = Low end of input voltage range.
V_{OH} = High end of output voltage range (FT-100 input).
V_{OL} = Low end of output voltage range (FT-100 input).

The DC voltage gain is determined by the resistors R1 and R2 in the following formula:

$$A_V = \frac{R2}{R1} \quad or \quad R2 = A_V * R1 \qquad \text{(EQUATION 2)}$$

We must select R1 and R2 to yield the desired gain. The exact values are not overly critical as long as the op-amp is not overstressed. In general, keeping R1 and R2 above 1K should always be safe. Remember, R1 sets the input impedance.

In order to offset the output voltage range, we must now determine the bias voltage (V_{BIAS}) by using the following formula:

$$V_{BIAS} = \left(\frac{(A_V * V_{IH}) - V_{OL}}{A_V + 1}\right) \qquad \text{(EQUATION 3)}$$

The bias voltage is determined by the voltage divider consisting of resistors R3, R4, and R5. Normally the bias voltage should be somewhat adjustable to allow for routine calibration.

Using these formulas, you should be able to scale your input voltage to the full scale input of the FT-100. Since the actual voltage being read by the FT-100 is a scaled voltage, and not the actual voltage input, you will need to scale the voltage reading in your program using the following formula:

$$V = V_{BIAS} - \left(\frac{(V_{IN} - V_{BIAS})}{A_V} \right) \qquad \text{(EQUATION 4)}$$

It may be best to look at some examples to get a better idea of how all these formulas work.

EXAMPLE #1

Given: Voltage range to be read = -2.0 to +2.0 volts
Want to use the FT-100 main unit analog input with a range of 0 to 10.0 volts

Using Equation 1, we find that our DC voltage gain is:

$$A_V = ABS\left(\frac{10.0 - 0.0}{2.0 - (-2.0)} \right) = 2.5$$

Where: $V_{IH} = 2.0$ $V_{OH} = 10.0$
$V_{IL} = -2.0$ $V_{OL} = 0.0$

We can now use Equation 2 to select R1 and R2 for a gain of 2.5. We will arbitrarily select R1 to be 100KΩ. This yields the following:

$$R2 = (2.5 * 100000) = 250000 = 250K\Omega$$

The bias voltage may be determined now, using Equation 3:

$$V_{BIAS} = \left(\frac{(2.5 * 2.0) + 0.0}{2.5 + 1} \right) = 1.43 \ volts$$

Now that we know all the critical elements, we can now build the circuit and fine tune any voltages as needed. The final circuit would look like:

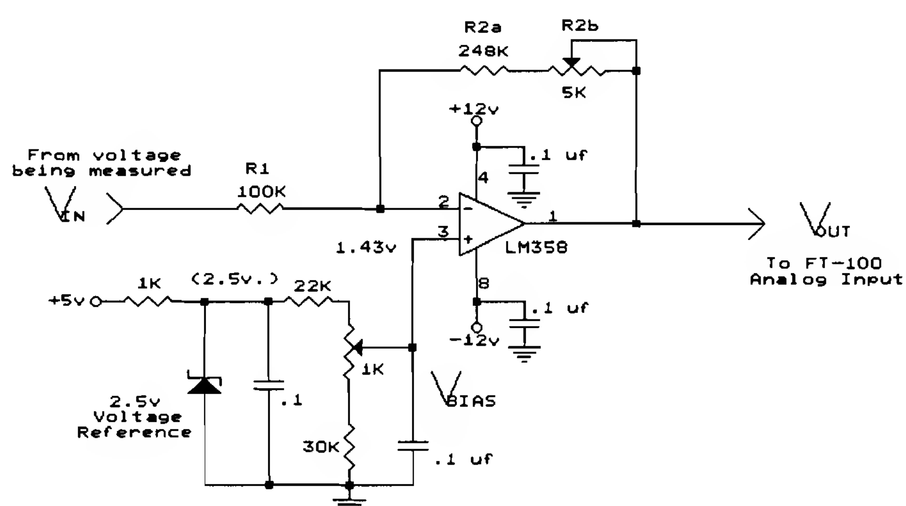


FIGURE 2 - EXAMPLE #1 FINAL SCHEMATIC

In order to read the correct voltages, the FT-100 must "correct" the scaled input voltage. Using Equation 4, the command would look like:

V = 1.43-((AA9-1.43)/2.5) (Assuming AA9 is used as the analog input line)

The variable V will now equal the exact voltage present at the input to your scaler circuit.

EXAMPLE 2

Given: Voltage range to be read: -30.0 to +20.0 volts.
Want to use an Expansion Module with input range of -5.0 to +5.0 volts.

From Equation 1 we get the DC gain:

$$A_V = \frac{5.0 - (-5.0)}{20.0 - (-30.0)} = 0.2$$

From Equation 2 we once again choose R1 to be 100KΩ. Solving for R2, we get:

$$R2 = (0.2 * 100000) = 20000 = 20K\Omega$$

From Equation 3, we get the bias voltage:

$$V_{BIAS} = \left(\frac{(0.2 * 20.0) + (-5.0)}{0.2 + 1} \right) = -0.833\ volts$$

The compensation formula that must be used in the program to "correct" the analog reading is:

V = -0.833-((AA9+0.833)/0.2) (Assuming AA9 is used as the analog input line)

Figure 3 shows the final circuit with all the selected values. Also note that there is an input buffer on this scaler circuit to increase the input impedance. The input buffer is completely optional.

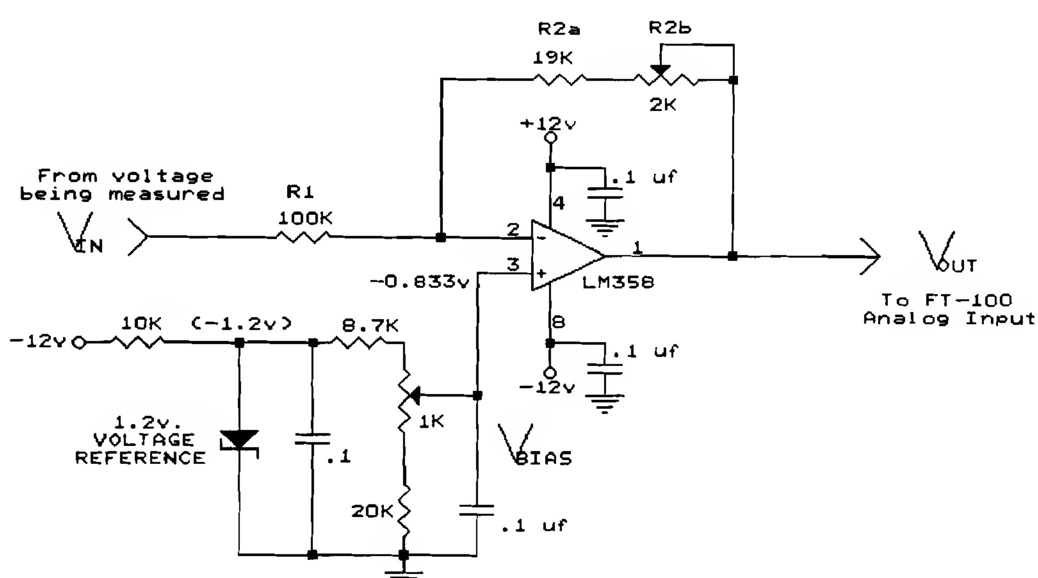


FIGURE 3 - EXAMPLE #2 FINAL SCHEMATIC

If you have several different voltage ranges that need to be monitored, or if you need high resolution over a larger input range, the circuit shown in figure 4 provides a very economical solution. By using an inexpensive analog multiplexer, such as the DG509, it is possible to have several voltage ranges using the same scaler circuit. The gain feedback resistor and the bias voltage are "switched" as desired. Two digital output lines from the FT-100 are used to automatically switch the multiplexer for the desired voltage range. This can be done within the test program, making it completely transparent to the operator.

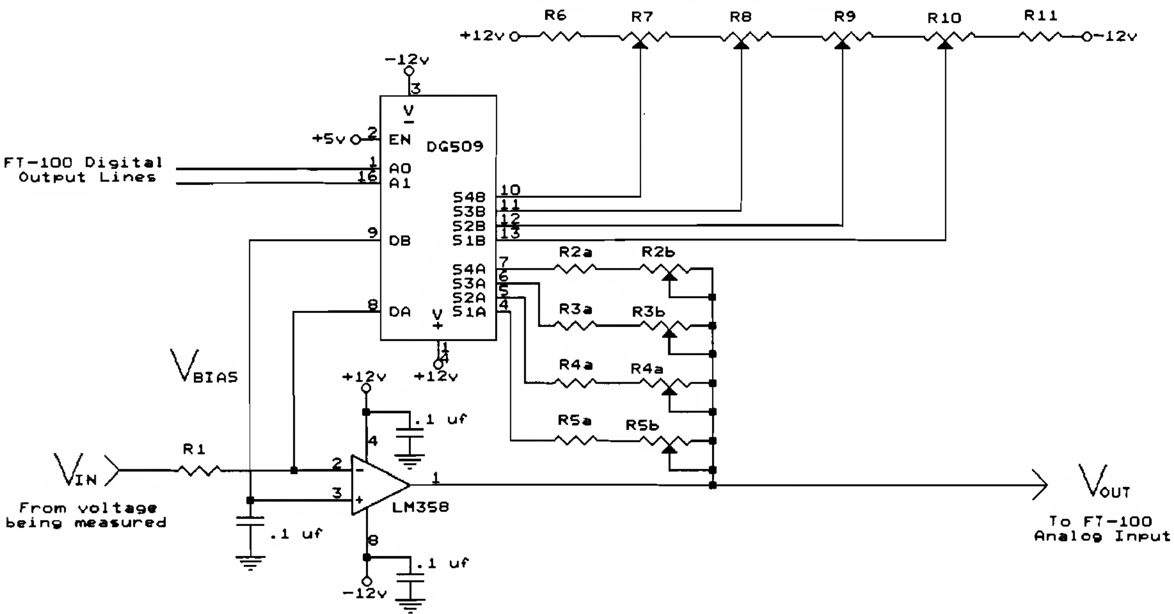


FIGURE 4 - MULTIPLE VOLTAGE RANGES AND GAINS

To extend the input range, and maintain a high resolution reading, divide the input range into several sub-ranges. For example, using the circuit shown in figure 4, we can select scaler values for an input range of 0-0.5 volts. Using the previous equations, this can be scaled to 0-10.0 volts output with a resolution of about 1.95 mv. To read a voltage above 0.5 volts, at the same resolution, we can keep the same amplifier gain but change the bias voltage to shift the output (using equation 3). The four input voltage ranges would be: 0-0.5, 0.5-1.0, 1.0-1.5, 1.5-2.0. We are now able to measure voltages from 0.0 to 2.0 volts at a resolution of 1.95 mv. The test program would need to read the input voltage at the lowest scale and determine if the reading is at the top end. If so, simply switch the DG509 analog multiplexer to the next higher voltage range. Continue this until the reading is within the input range.

The DG509 analog multiplexer allows up to 4 different voltage ranges and 4 different gains, but a DG507 increases this number to 8. If the gain can remain constant, as in our previous example, a DG506 will allow up to 16 different voltage ranges! These are all very inexpensive integrated circuits, and are readily available from several sources.

Using these methods, it is very easy to monitor almost any voltage range - and very inexpensively! If you have particular needs and aren't sure how to go about designing your scaler circuit, or if you aren't sure where to purchase or how to use analog multiplexers, give Y-tek a call. An Applications Engineer will be glad to help you.

V-tek, Inc. --- *APPLICATION NOTE* ---

App. Note#: AN-0100-07 Product: FT-100 Date Issued: May, 1995

Subject: STORING WAVEFORMS IN THE FT-100

Many times it would be beneficial to capture and store a waveform (or continuously changing DC voltage) to be viewed or analyzed later, either by the test program or the operator. A digital storage oscilloscope (DSO) can accomplish this very easy. With the FT-100 controlling the scope through the RS-232 port, waveforms may be downloaded and analyzed as needed. The only problem with this scenario is that most DSO's are relatively expensive and aren't always available when needed.

This application note describes a simple FT-100 software routine that may be used to capture a waveform and optionally store it on disk. The waveform is stored in an ASCII format that is recognized by most graphing, spreadsheet, and data analysis programs. Once captured, the waveform can be analyzed by the FT-100 just as easily. This routine is very flexible and can be easily modified for each given application.

The FT-100 could never compete with a high-speed DSO, but for some limited applications, the ability to store and analyze a waveform, coupled with all the other features of the FT-100, gives you a very inexpensive method of data analysis. For many tasks, a high-priced DSO cannot be justified, but the ability to record a waveform could make the test job a lot easier and more thorough.

Possible Applications

- Low frequency audio analysis.
- DC pulse-width measurements.
- RC time-constants - This could be used to roughly read a capacitor or resistor value.
- Waveform comparisons.
- Power supply rise and fall time.
- Low-frequency distortion analysis.
- Switch "bounce" characteristics.
- "Peak" voltage measurements.

Features

- 12-bit analog input.
- Any expansion module analog input line may be used for the waveform input.
- Triggering may be from any of the following sources:
  - Analog input line (trigger voltage may be set).
  - Digital input line (can be any digital input line, including those in the main unit).
  - Trigger polarity may be positive or negative (for analog or digital trigger inputs).
- Number of samples recorded may be specified, and is limited only by the amount of available memory in the FT-100. (The routine indicates the available memory.)
- Time between each sample is programmable.
- Waveform may be stored on a floppy disk. Data is stored in ASCII format that is readable by most data analysis programs and spreadsheets.
- Waveform may be stored in an array for analysis by the FT-100 program.
- Normally, several thousand data points can be stored (up to over 60,000!).

Limitations

Because the FT-100 was not designed specifically to be a digital oscilloscope, there are some limitations that you must keep in mind before, and during, implementation of this routine:

- Only analog I/O Expansion Modules may be used for the waveform input. The main unit cannot be used.
- The smallest sample speed allowed is 42 μsec. per sample.
- The time between each sample is always in increments of around 2.8 μsec.
- During waveform recording, the FT-100 cannot perform other tasks.

How the Waveform Capture Routine Works

The real work is done inside the assembly subroutine to maximize speed and efficiency. When keying in the assembly routine data statements, *be very careful!* It is very easy to key in an incorrect data number. After the program loads in the assembly subroutine, the parameters for the waveform capture routine are set. Your particular application will most likely determine how and when these parameters should be set. The parameter information is passed to the assembly routine by placing the information in a given section of RAM (through POKE commands). After all the parameter information is placed in RAM, the assembly subroutine can be "called" at any time. The assembly subroutine stores the waveform in memory and returns to the main program.

After the waveform data hs been stored in memory, it can then be stored to a disk file, sent out the serial port, or saved in an array for processing and analysis within the FT-100. In order to save the information in an array, place the following set of commands *after* the SYS command that calls the assembly subroutine (Note: If an array is used to store the data, it must be dimensioned at the beginning of the program.):

```
BEG_ADDR=DATA_SEG*16                                  'Address in RAM of the beginning of the data
S1 = F_SCALE/((1+(RES*15))*256):S2 = REF*(F_SCALE/2)         'Voltage scaler multipliers
FOR X=0 TO (SAMPLES-1)
A(X) = S1 * (PEEK(I+BEG_ADDR)+PEEK(I+BEG_ADDR+1)*256) - S2
INC I : NEXT
```

There are several commands that calculate and process information to be placed in the FT-100 operating system. These appear confusing, and probably don't make much sense, but they are necessary for the assembly routine to operate most efficiently.

When keying in this program, the comments may be omitted as you see fit. They are there only to help clarify the program operation.

```
WAVE_1                    'Program Name
'
   PROG_LEN= 170                          'Number of bytes in assembly program
'
'--- Test for enough memory to load in assembly routine ---
   LAB_BEG=PEEK 45+(PEEK 46*256)+((PEEK 47+(PEEK 48*256))*16)
   STR_BEG=PEEK 59+(PEEK 60*256)+((PEEK 61+(PEEK 62*256))*16)
   MEM_FREE = LAB_BEG - STR_BEG-16                         'Total available memory
   IF PROG_LEN>MEM_FREE THEN BEEP:CLS:DISP:DISP "INSUFFICIENT":DISP "MEMORY!":END
'
   MEM_DATA = MEM_FREE - PROG_LEN                  'Memory available for data storage
'
'--- Read in assembly routine & store in memory ---
   PROG_SEG=INT (STR_BEG/16)+1:PROG_OFF=0
   ADDR=PROG_SEG*16
   FOR X=0 TO PROG_LEN:READ D:POKE (X+ADDR),D:NEXT              'Load assembly program
'
'---------------- Assembly Program DATA statements ---------------------
     DATA 1EH,2BH,0D2H,8EH,0DAH,0BEH,74H,1
     DATA 0FFH,34H,46H,46H,8BH,0CH,46H,46H
     DATA 8BH,1CH,46H,46H,8AH,14H,46H,8BH
     DATA 2CH,46H,46H,8AH,24H,46H,8BH,3CH
     DATA 8EH,0DFH,2BH,0FFH,5EH,8AH,0C7H,0EEH
     DATA 0B0H,1EH,0FEH,0C8H,75H,0FCH,0F6H,0C3H
     DATA 1,75H,29H,0F6H,0C3H,2,75H,45H
     DATA 0E8H,11H,0,89H,5,0E2H,2,1FH
     DATA 0CBH,47H,47H,8BH,0C6H,2DH,1,0
     DATA 75H,0FBH,0EBH,0ECH,8AH,0C7H,0EEH,42H
     DATA 0B0H,10,2CH,1,75H,0FCH,0ECH,8AH
     DATA 0E0H,4AH,0ECH,0C3H,51H,0E8H,0ECH,0FFH
     DATA 8BH,0C8H,0E8H,0E7H,0FFH,0F6H,0C3H,80H
     DATA 74H,0BH,3BH,0C5H,73H,0F2H,2BH,0E9H
     DATA 73H,0EEH,59H,0EBH,0C6H,3BH,0C5H,76H
     DATA 0E7H,3BH,0CDH,0EBH,0F3H,51H,52H,8AH
     DATA 0D4H,8AH,0CCH,80H,0E1H,7,0B4H,1
     DATA 0D2H,0E4H,0B1H,4,0D2H,0EAH,0ECH,22H
     DATA 0C4H,8AH,0E8H,0ECH,22H,0C4H,3AH,0C5H
     DATA 74H,0F9H,8AH,0E8H,0F6H,0C3H,80H,75H
     DATA 2,32H,0C4H,3CH,0,75H,0ECH,5AH
     DATA 59H,0EBH,8DH
'------------------------------------------------------------------------------
'
'----- Set variables before calling the waveform capture subroutine -----
' (NOTE:  There are no checks for correct data form.  All alpha characters
'      MUST be capital letters.  This should be no problem since the
'      input information is created in your program.)
'
     ANA_IN$ = "AB6"            'Analog input line used to input waveform
     SAMPLES = 1000             '# of sample of data to store
     TIME_DEL = 100             'Time delay between each sample (µsec)
     TRIG$ = "AB6"              'Trigger source -(analog, digital, "" = none)
     TRIG_POL = 0               'Trigger polarity (0=pos, 1=neg)
     TRIG_V = 2.5               'DC trigger voltage (volts)  <optional>
'
'
'---- Place information in memory locations for waveform capture routine to use ----
'
' Set the location for the beginning of data storage
     DATA_SEG=INT(PROG_SEG+(PROG_LEN/16)+1)
     POKE 17EH,(DATA_SEG-(INT(DATA_SEG/256)*256))         '(LSB) Storage segment
     POKE 17FH,DATA_SEG/256                               '(MSB)
'
```

```
   POKE 176H,(SAMPLES-(INT(SAMPLES/256)*256))              '(LSB) # of samples
   POKE 177H,SAMPLES/256                                   '(MSB)
'
   COUNT = INT((TIME_DEL-42.2)/2.79611+0.5)+1
   POKE 174H,(COUNT-(INT(COUNT/256)*256))                  '(LSB) Delay between each sample
   POKE 175H,COUNT/256                                     '(MSB)
'
   ANA_MOD = ASC(MID$(ANA_IN$,2,1))-41H              'ANALOG MODULE# (0-7)
   DL=PEEK ((ANA_MOD-1)*2+14H)                       'DL, DH = Module configuration data
   DH=PEEK ((ANA_MOD-1)*2+15H)
   RES = (DL AND 80H)/80H                            'Analog Resolution (0=8-BIT, 1=12-BIT)
   REF = (DL AND 40H)/40H                            'Analog Reference (0=0V, 1=+/-V.)
   F_SCALE = 5+((DH AND 8)/8*5)                      'Full-scale analog voltage
'
   POKE 17AH,(ANA_MOD*16) OR (DL AND 0FH)                                 'Set Analog in/out address
   POKE 179H,(VAL(RIGHT$(ANA_IN$,(LEN ANA_IN$-2)))-(2*(DH AND 7)))-1  'Analog line#
'
' ----- Set up the trigger information -----
  TRIG=TRIG_V
   IF REF=1 THEN TRIG=TRIG+F_SCALE:F_SCALE=F_SCALE*2
   X = ((256*((RES*15)+1))/F_SCALE)*TRIG
   POKE 17BH,(X-(INT (X/256)*256))                         '(LSB) Trigger Voltage
   POKE 17CH,X/256                                         '(MSB)
'
   TRIG = 0
   IF TRIG$="" THEN GOTO L0                                'No trigger
   IF TRIG$=ANA_IN$ THEN TRIG=TRIG OR 1:GOTO L0            'Analog trigger
'
   TRIG=TRIG OR (ASC(MID$(TRIG$,2,1))-41H)16)
   LINE_NUM = VAL(RIGHT$(TRIG$,(LEN(TRIG$)-2)))            'Digital trigger line#
   X = ((LINE_NUM-1)/8)-(DL AND 0FH)
   PORT = INT X                                            'Ttrigger port#
   LINE = (X-INT X)*8                                      'line# in the digital port
   POKE 17DH,(PORT*16 OR LINE)                             'Digital trigger port# & line#
'
L0:  TRIG=TRIG+(TRIG_POL*80H)                              'Trigger polarity
   POKE 178H,(TRIG OR 2)                                   'Trigger source
'
'
'-------- Test for enough memory to store waveform ----------
  IF MEM_DATA < (RES+1)*SAMPLES THEN BEEP:CLS:DISP:DISP "INSUFFICIENT":DISP "MEMORY!":END
'
  SYS PROG_SEG,PROG_OFF                              'Call the assembly subroutine
'
' At this point the waveform is in memory.  If the waveform is to be saved to the disk,
' use the following routine:
'
   BEG_ADDR=DATA_SEG*16                              'Address in RAM of the beginning of the data
   S1 = F_SCALE/((1+(RES*15))*256): S2 = REF*(F_SCALE/2)   'Voltage scaler multipliers
   OPEN "TRACE",OUT                                        'Open the disk file
   FOR I=0 TO (SAMPLES-1)
   STORE S1 * (PEEK(I+BEG_ADDR)+PEEK(I+BEG_ADDR+1)*256) - S2
   INC I : NEXT
   CLOSE                                                   'Close the disk file
'
/
```

V-tek, Inc. --- *APPLICATION NOTE* ---

App. Note#: AN-0100-08 Product: FT-100 Date Issued: June, 1995

Subject: LOGIC CAPTURE AND STORE IN THE FT-100

This application note describes a very simple method of capturing and storing eight lines of digital information, much like a digital logic analyzer. No additional hardware is necessary, and to keep the routine simple, only the main FT-100 unit is used for the digital input as well as the trigger source. Once the digital information is captured and stored in memory, you are free to treat the data as your needs dictate. The data may be stored on a disk file, printed out, downloaded to a computer, or analyzed for specific features and later discarded.

Features

- Uses no additional hardware. Main FT-100 unit is used for all capture and trigger functions.
- Several triggering options are available:
  - No triggering (free running)
  - Digital Input Line (main unit only)
  - Logic Word matching
  - Externally gated data
  - Trigger polarity may be positive- or negative-going
- Number of samples recorded may be specified, and is limited only by the amount of available memory in the FT-100. (The routine indicates the available memory.)
- Time between each sample is programmable.
- Data may be stored on a floppy disk
- Data may be stored in an array for analysis by the FT-100 program.
- Normally, several thousand data samples can be stored (up to over 60,000!).

Limitations

- Only 8 lines of data are captured.
- The only Digital Input Port used to capture data is PA10.
- The smallest sample speed allowed is 8 µsec. per sample.
- The time between each sample is always in increments of around 2.8 µsec.
- If data is externally gated in, the recycling time is about 11.6 µsec.
- During data capture, the FT-100 cannot perform other tasks. The FT-100 will remain in the assembly routine until all the data is captured.
- All timing is based on the microprocessor crystal, and may not be accurate enough for some very high-accuracy applications.
- Externally gated data relies on a software trigger. This causes a small delay between the trigger and the captured data (usually less than one µsec.).
- All internal interrupts are enabled during data capture. If this is a problem, disable interrupts using the DINT command prior to calling the assembly subroutine.

To maximize speed, the data is captured and stored by an assembly subroutine. The accompanying program READ's and POKE's each assembly byte into the FT-100 memory. Triggering is also done inside the assembly subroutine. All input information is passed to the assembly subroutine through specified, and fixed, RAM locations. The data is captured and placed in RAM beginning at the RAM location indicated by BEG_ADDR. All data is stored in byte format (binary) as input from the Digital Input Port (PA10), and may be retrieved by using the PEEK command.

Once the assembly routine has been "poked" into memory and the input parameters "poked" into their designated RAM locations, the assembly subroutine may be called by using the SYS command. Any time you wish to capture and store digital data, simply call the subroutine. The input parameters, as well as the assembly program, will remain in RAM and will not be changed as long as the main program remains running. If the program is stopped or ends, the information will still remain in RAM, and may be accessed directly (in the Direct Command mode). If a different program is loaded into the FT-100, anything previously in RAM memory will not be valid.

- IMPORTANT -
This program does not test any parameters for proper format.
Your program should be sure all parameters are valid
prior to calling the assembly subroutine!

CAPTURING DIGITAL DATA

Once the assembly subroutine is poked into memory, it may be called anytime, using the SYS command. The actual memory location of the assembly subroutine is indicated by the variables PROG_SEG and PROG_OFF. Prior to calling the assembly subroutine, the input parameters must be established and passed to the subroutine through the designated RAM locations. Once the parameters are placed in RAM, they will remain valid and need not be reset unless your program changes a value. Following are the input parameters and a description of each:

TIME_DIV - Delay time between each sample, measured in microseconds (μs.). This variable may be any value between 8 and 65535. If the external trigger line is used to latch each sample, TIME_DIV is not used.

SAMPLES - Number of samples of data to be captured and stored. This variable may be any value between 1 and 65535.

TRIG$ - Trigger source. This is a *string variable* that tells the program which trigger source to use. There are several triggering options available:

No trigger - Free running. Captures data instantly with no waiting. TRIG$ = "" to indicate no triggering.

Digital Input Line - A transition on the designated digital input line will trigger the data capture. Any digital input line in the main FT-100 unit may be used (including PA10). TRIG$ = "LAxx". (LA41 - LA80). (NOTE: TRIG$ cannot be of the form: "LAx,y")

Logic Word - May be any 8-bit word. The digital input port PA10 is constantly sampled, looking for the designated "Logic Word". When a match is found, the capture routine is triggered. TRIG$ may be set to any string value that is equivalent to a numerical value of 0 to 255.
Examples: TRIG$ = "134", TRIG$ = "0FCH", TRIG$ = "10110010B"
If a variable is used: TRIG$ = STR$ (VAR1)

External Latch - Latches one sample (8 bits) at each transition of the external trigger line. The main unit external trigger line is used to gate in one logic word of digital information. TRIG$ = "EXT".
(See "LIMITATIONS" for more information about external triggering)

TRIG_POL - Trigger polarity. Set to zero (Ø) for trigger on positive-going transition. Set to one (1) for negative-going transition. (Not used for "Logic Word" triggering).

ACCESSING THE DATA

The digital data is stored in RAM beginning at the location indicated by the variable BEG_ADDR. The data may be accessed by using the PEEK command. Data is stored in binary format, just as it is read from Digital Input Port PA10. The program listing shows a method of storing the data on a disk file. What you choose to do with the data after it is captured is completely up to you. The data will remain valid in RAM until more data is captured.

When keying in this program, the comments may be omitted. They are there only for clarification. Also, use care when entering the DATA statements (assembly subroutine). They must be <u>exact</u> for the subroutine to function properly.

```
LOGIC_1                'Program Name
'
   PROG_LEN= 93H        'Number of bytes in assembly subroutine
'
' ---- Find location in RAM to place assembly routine ---
   LAB_BEG=PEEK 45+(PEEK 46*256)+((PEEK 47+(PEEK 48*256))*16)
   STR_BEG=PEEK 59+(PEEK 60*256)+((PEEK 61+(PEEK 62*256))*16)
   MEM_FREE = LAB_BEG-STR_BEG-16           'Amount of free memory available for assembly program
   PROG_SEG=INT (STR_BEG/16)+1
   PROG_OFF=0
   ADDR=PROG_SEG*16
'
'--- Test for enough memory ---
   IF PROG_LEN>MEM_FREE THEN BEEP:CLS:DISP "INSUFFICIENT":DISP "MEMORY!":END
'
   FOR I=0 TO PROG_LEN:READ X:POKE (I+ADDR),X:NEXT            'Load assembly program
'
'---------------------------- Assembly Program ----------------------------
   DATA 006H,02BH,0FFH,0FCH,0BEH,077H,001H,08AH
   DATA 014H,046H,08BH,01CH,046H,046H,08BH,02CH
   DATA 046H,046H,08BH,00CH,046H,046H,0FFH,034H
   DATA 007H,0F6H,0C7H,040H,074H,00EH,0F6H,0C7H
   DATA 020H,075H,044H,0E4H,004H,03AH,0C3H,075H
   DATA 0FAH,0EBH,02EH,090H,0F6H,0C7H,020H,074H
   DATA 026H,051H,02AH,0F6H,08AH,0CFH,080H,0E1H
   DATA 007H,0B4H,001H,0D2H,0E4H,0ECH,022H,0C4H
   DATA 08AH,0E8H,0ECH,022H,0C4H,03AH,0C5H,074H
   DATA 0F9H,08AH,0E8H,0F6H,0C7H,080H,075H,002H
   DATA 032H,0C4H,03CH,000H,075H,0ECH,059H,0E4H
   DATA 004H,0AAH,0E2H,002H,007H,0CBH,08BH,0C5H
   DATA 02DH,001H,000H,075H,0FBH,0EBH,0F0H,051H
   DATA 08AH,026H,080H,000H,080H,0E7H,080H,0B1H
   DATA 004H,0D2H,0EFH,059H,00AH,0E7H,080H,0CCH
   DATA 004H,088H,026H,080H,000H,08AH,0C4H,024H
   DATA 0FBH,0E6H,005H,08AH,0C4H,0E6H,005H,0E4H
   DATA 006H,0D0H,0E0H,073H,0FAH,0E4H,004H,0AAH
   DATA 0E2H,0EBH,0EBH,0C8H
'-------------------------------------------------------------------------------------
'
'----- Set Input parameters before calling the Digital Capture subroutine -----
'
    TRIG$ = "LA41"                  'Trigger line ("EXT"=external, ""=none)
    SAMPLES = 1000                  'Number of samples of data to store
    TIME_DIV = 150                  'Time delay between each sample (µsec)
    TRIG_POL = 0                    'Trigger polarity (0=pos, 1=neg)
```

```
'
'----- Test for enough memory to store waveform -----
   MEM_DATA = MEM_FREE - PROG_LEN                          'Memory available for data storage
   IF MEM_DATA<SAMPLES THEN CLS:BEEP:DISP "INSUFFICIENT":DISP "MEMORY!":END
'
'---- Place Input parameters in memory locations for Digital Capture subroutine to use ----
'
    DATA_SEG=INT(PROG_SEG+(PROG_LEN/16)+1)                    'Location for data storage
    POKE 17EH,(DATA_SEG-(INT(DATA_SEG/256)*256))              '(LSB) STORAGE SEG.
    POKE 17FH,DATA_SEG/256                                    '(MSB)
'
    POKE 17CH,(SAMPLES-(INT(SAMPLES/256)*256))                '(LSB) Number of samples
    POKE 17DH,SAMPLES/256                               '(MSB)
'
   COUNT=INT((TIME_DIV-8.1)/2.8114+0.5)+1
   POKE 17AH,(COUNT-(INT(COUNT/256)*256))                     '(LSB) Sample delay
   POKE 17BH,COUNT/256                                        '(MSB)
'
'----- TRIGGER INFO -----
   IF TRIG$="" THEN POKE 179H,0:GOTO L0                              'No trigger
'
   IF TRIG$="EXT" THEN POKE 179H,(60H OR (TRIG_POL*80H)):GOTO L0     'External trigger
'
   IF LEFT$(TRIG$,1)="L" THEN GOTO L1                                'Line trigger?
   POKE 179H,40H:POKE 178H,VAL TRIG$                                 'Word trigger
   GOTO L0
'
' ----- LINE TRIGGER -----
L1:  TR_LINE = VAL(RIGHT$(TRIG$,(LEN(TRIG$)-2)))               'Trigger line#
   TR_PORT = INT ((TR_LINE-1)/8)                               'Trigger Port#
   POKE 177H,TR_PORT-5
   X=(TR_LINE-1)/8
   POKE 179H,((X-INT X)*8) OR 20H OR (TRIG_POL*80H)
'
L0:  SYS PROG_SEG,PROG_OFF                      'Call assembly routine to capture data
'
   BEG_ADDR=DATA_SEG*16                  'Location of the beginning of the stored data
'
' At this point the captured digital information is in memory (in binary format) beginning at memory
' location BEG_ADDR.  If the waveform is to be saved to a disk file, use the following routine:
'
'----- Store captured digital information to a disk file (optional) -----
   OPEN "DIGITAL.DAT",OUT
   FOR I=0 TO (SAMPLES-1)          '(Note:  If SAMPLES is larger than 32767, two loops must be used)
   STORE PEEK(I+BEG_ADDR)
   NEXT
   CLOSE
/
```

V-tek, Inc. --- *APPLICATION NOTE* ---

App. Note#: AN-0100-09 Product: FT-100 Date Issued: January, 1996

Subject: DATA ACQUISITION WITH EXTECH DIGITAL MULTI-METER

The EXTECH Model 383273 Digital Multi-meter has a built-in RS-232 interface, and can easily communicate with the FT-100. The meter is relatively inexpensive (less than $200), and provides several data monitoring functions that can expand the capabilities of the FT-100. In addition to the typical DMM features, there are also several optional probes that allow the monitoring of several other types of inputs, such as: humidity, light intensity, wind speed, etc. Although the meter was originally designed to interface with a PC, this application note describes the hardware and software necessary for interfacing with the FT-100.

The EXTECH 383273 is a 3½ digit digital multi-meter with the following measurement capabilities:

- Voltage (AC & DC)
- Current (AC & DC)
- Resistance
- Capacitance
- Frequency
- Diode and Continuity
- Temperature

The meter has its own RS-232 interconnect cable; however, to interface with the FT-100, this cable must be altered. The schematic below shows the necessary cable interconnections:

```
                DB9              DB25
                 2 >---------------<3
                 3 >---------------<2
  Meter          4 >---------------<9     FT-100
                 5 >---------------<7
                 7 >-x           +--<4
                                 +--<5
```

The EXTECH meter always requires a "high" on pin 4 (DTR) to send data. The above cable connects this line to pin 9 from the FT-100. Pin 9 is one of the two lines used for serial port selection in the 1041 Serial Port Expander. Because of this, the meter must be addressed as either port #2 or #4. Note: The 1041 does not have a "select" voltage on pin 9. If you need to interface with the meter through a 1041, please contact the factory.

The program shown can be used to receive all types of readings from the meter. Since most applications would not require all the functions of the meter, you may omit the program lines that are not necessary for your particular function. In particular, unused "IF" statements used for function selection, may be omitted.

HOW THE PROGRAM WORKS

The meter will immediately send 5 bytes of data whenever it receives a "space" character. The first byte is always 02, and the last byte is always 03. These framing bytes are used to determine proper communications. The third byte is the function and range byte, and is used in the program to select the proper "units" and "multiplier". The third and fourth bytes contain the actual reading. This information is embedded within the bits of these numbers, so the program extracts the data from the bytes. The multiplier is used to scale the meter reading to the basic units for each type of reading (for example, capacitance is scaled to farads). The variable "VALUE" is the final value for the reading, in the basic units.

The program reads in the 5 bytes and stores them as string variables. Because these bytes can be any value from 00 to 0FFH, and therefore cannot always be represented as ASCII text, we must extract the actual value from the string variable table. In order to know exactly <u>where</u> in the table each string variable is located, it is <u>very important</u> that the <u>first</u> reference to string variables folllows the order: A$, B$, C$, D$, E$. To simplify this requirement, you could place the following command at the beginning of your program:

```
A$="": B$="": C$="": D$="": E$=""
```

This will insure the placement of the string variables within the table.

Although somewhat lengthy, this program is relatively simple, and you should have very little trouble following the flow and adapting it to your particular situation. If you have any trouble, or have a unique interface situation, please contact the factory. There is always an application engineer ready to assist.

```
EXTECH          'Program Name
'
        COM 9600,,,,0,,0              'Set baud to 9600 & disable Direct Command Mode & Echo
L0:     ? @2;" ";:DELAY 10            'Send "space" character to meter, & wait small delay for return
        GET @2;A$,B$,C$,D$,E$                          'Read 5 bytes from meter
        IF A$<>CHR$ 2 OR E$<>CHR$ 3 THEN GOTO L1       'Test framing bytes
        CLS
'
'------ Extract readings from STRING VARIABLE table ------
        PTR=PEEK 59+(PEEK 60*256)+((PEEK 61+(PEEK 62*256))*16)-4
        FUNCT=(PEEK PTR AND 1)*0FFH AND (PEEK (PTR-1))          'Extract Function byte
        PTR=PTR-2:IF FUNCT<>0 THEN DEC PTR
        VAL1=(PEEK PTR AND 1)*0FFH AND (PEEK (PTR-1))           'Extract #1 Data byte
        PTR=PTR-2:IF VAL1<>0 THEN DEC PTR
        VAL2=(PEEK PTR AND 1)*0FFH AND (PEEK (PTR-1))           'Extract #2 Data byte
'
'------ Test for invalid or overload readings ------
        IF (VAL1 AND 1FH)=3FH THEN CLS:GOTO L0
        IF (VAL1 AND 1FH)=0FH THEN CLS:CURSOR 2,5:DISP "+ OVERLOAD":GOTO L1
        IF (VAL1 AND 1FH)=0EH THEN CLS:CURSOR 2,5:DISP "- OVERLOAD":GOTO L1
'
'------ Determine Function and Value Multiplier of Meter Setting------
        IF FUNCT=0 THEN F$="V DC":M=0.0001                  '200mv DC scale
        IF FUNCT=1 THEN F$="V DC":M=0.001                   '2v DC scale
        IF FUNCT=2 THEN F$="V DC":M=0.01                    '20v DC scale
        IF FUNCT=3 THEN F$="V DC":M=0.1                     '200v DC scale
        IF FUNCT=4 THEN F$="V DC":M=1                       '1000v DC scale
        IF FUNCT=5 THEN F$="Hz":M=1                         'Frequency
        IF FUNCT=6 THEN F$="V DC <CONT/DIODE>":M=0.001      'Diode/Continuity scale
        IF FUNCT=8 THEN F$="Ohms":M=0.1                     '200 Ω Resistance scale
        IF FUNCT=9 THEN F$="Ohms":M=1                       '2K Ω Resistance scale
        IF FUNCT=10 THEN F$="Ohms":M=10                     '20K Ω Resistance
        IF FUNCT=12 THEN F$="Ohms":M=100                    '200K Ω Resistance
        IF FUNCT=16 THEN F$="Ohms":M=1000                   '2M Ω Resistance scale
        IF FUNCT=17 THEN F$="Ohms":M=10000                  '20M Ω Resistance scale
        IF FUNCT=18 THEN F$="CAP (F)":M=1E-8                '20 F Capacitance scale
        IF FUNCT=20 THEN F$="CAP (F)":M=1E-9                '2 F Capacitance scale
        IF FUNCT=24 THEN F$="CAP (F)":M=1E-10               '200nF Capacitance scale
        IF FUNCT=32 THEN F$="CAP (F)":M=1E-12               '2000pF Capacitance scale
        IF FUNCT=33 THEN F$="A DC":M=0.01                   '20 Amp DC scale
```

```
        IF FUNCT=34 THEN F$="A DC":M=0.0001                '200 mA DC scale
        IF FUNCT=36 THEN F$="A DC":M=1E-5                  '20 mA DC scale
        IF FUNCT=40 THEN F$="A DC":M=1E-6                  '2 mA DC scale
        IF FUNCT=48 THEN F$="A DC":M=1E-7                  '200  A DC scale
        IF FUNCT=64 THEN F$="DEG F":M=0.1                  'Temperature 200° F
        IF FUNCT=65 THEN F$="DEG F":M=1                    'Temperature 2000° F
        IF FUNCT=66 THEN F$="DEG C":M=0.1                  'Temperature 200° C
        IF FUNCT=68 THEN F$="DEG C":M=1                    'Temperature 2000° C
        IF FUNCT=128 THEN F$="V AC":M=0.0001               '200mv AC scale
        IF FUNCT=129 THEN F$="V AC":M=0.001                '2v AC scale
        IF FUNCT=130 THEN F$="V AC":M=0.01                 '20v AC scale
        IF FUNCT=131 THEN F$="V AC":M=0.1                  '200v AC scale
        IF FUNCT=132 THEN F$="V AC":M=1                    '750v AC scale
        IF FUNCT=161 THEN F$="A AC":M=0.01                 '20 Amp AC scale
        IF FUNCT=162 THEN F$="A AC":M=0.0001               '200 mA AC scale
        IF FUNCT=164 THEN F$="A AC":M=1E-5                 '20 mA AC scale
        IF FUNCT=168 THEN F$="A AC":M=1E-6                 '2 mA AC scale
        IF FUNCT=176 THEN F$="A AC":M=1E-7                 '200  A AC scale
        IF FUNCT=255 THEN F$="*** HOLD ***"        'HOLD reading (HOLD button is pressed)
'
        POL=1-(1-(VAL1 AND 1))*2                           'Polarity
        IF FUNCT=5 THEN POL=1:IF (VAL1 AND 1)=1 THEN M=1E4 'Frequency Range
'
'--- Extract meter reading from data bytes ---
        NUM=(VAL2 AND 3CH)/4:CALL XPOSE:VALUE=Q            'LSD of result
        NUM=(VAL2 AND 3)*4+(VAL1 AND 0C0H)/64:CALL XPOSE
        VALUE=Q*10+VALUE
        NUM=(VAL1 AND 3CH)/4:CALL XPOSE:VALUE=Q*100+VALUE
        VALUE=(VAL1 AND 2)*500+VALUE                       'MSD of result
'
        IF (VAL2 AND 40H)=1 THEN VALUE=VALUE*100           'Decimal position
        IF (VAL2 AND 80H)=1 THEN VALUE=VALUE*10
        VALUE=VALUE*M*POL                          'Scale value to basic units & add polarity
'
        CURSOR 2,1:DISP #D;VALUE;" ";F$            'Display meter reading and units on LCD
'
L1:     DELAY 500:IF FUNCT=5 THEN DELAY 500        'Delay ½ sec. (If frequency, delay 1 sec.)
        GOTO L0
'
'
'******* SUBROUTINE TO TRANSPOSE BITS *******
'  Input: NUM = number to transpose
' Output: Q = result
'
XPOSE:  Q=0:FOR I=1 TO 4
          SHL Q:Q=Q+(NUM AND 1):SHR NUM
          NEXT:RETURN
/
```

V-tek, Inc. --- *APPLICATION NOTE* ---

App. Note#: AN-0100-10 Product: FT-100 Date Issued: December, 1996

Subject: FREQUENCY COUNTER PROGRAM

To functionally test some circuits, a need often arises to measure the frequency of a signal. External frequency counters may be used, but can be expensive and may offer more capabilities and/or accuracy than actually needed. If your signal is below about 40 KHz, the following software program could be good enough for your application. Plus, no extra equipment is needed!

This frequency counter program has some limitations that should be considered:

- Timing is dependent upon the FT-100 microprocessor crystal. For critical measurements, this may not be good enough.
- Only digital inputs in the main unit (module A) can be used.
- The count duration is fixed at about 0.65 seconds.

How the program works

The actual frequency counting is done by an assembly subroutine. Parameter values are passed to/from the assembly routine through designated RAM locations. Once the main test program loads the necessary parameters into RAM, they do not normally need to be reloaded again unless the program is stopped or restarted.

The assembly subroutine monitors the input line and counts the number of positive-going pulses for the count duration of about 0.65 seconds. It will then place the count value in the RAM locations and return to the main program. The main program then takes the raw count value and converts it into frequency (Hz) by multiplying it by a constant (CONST).

Calibration

Because of the tolerance inherent in all crystals, this routine should be calibrated using a good frequency counter or signal generator. To calibrate, input a known-frequency signal and record the CNT value returned from the assembly routine. The CONST value is derived from the following:

CONST = FREQUENCY / CNT

Once CONST is found, place this value in your program to be multiplied times the CNT value returned from the assembly subroutine.

The Frequency Counter Program

This frequency counter program simply displays, on the LCD, the frequency of the incoming signal. Your program will most likely need to handle the frequency value differently.

Be very careful when copying the program, especially the assembly DATA statements! Comments may be discarded. They are included only for clarity.

```
FREQ1                  'Program Name
'
    PROG_LEN = 50          '# of bytes in assembly program
'
'--- Test for enough memory ---
   LAB_BEG=PEEK 45+(PEEK 46*256)+((PEEK 47+(PEEK 48*256))*16)
   STR_BEG=PEEK 59+(PEEK 60*256)+((PEEK 61+(PEEK 62*256))*16)
   IF LAB_BEG<(STR_BEG+PROG_LEN)+16 THEN BEEP:CLS:DISP:DISP "NOT ENOUGH":DISP "  MEMORY!":END
'
   PROG_SEG=INT (STR_BEG/16)+1
   ADDR=PROG_SEG*16:FOR I=0 TO PROG_LEN
   READ X:POKE (I+ADDR),X:NEXT           'Load in assembly program
'
'------------------- Assembly Program -------------------
    DATA 0BEH,7AH,1,2BH,0FFH,8AH,14H,2AH
    DATA 0F6H,46H,8AH,3CH,46H,8BH,0CH,46H
    DATA 46H,8AH,0E7H,0ECH,22H,0C7H,3AH,0C4H
    DATA 74H,0BH,3CH,0,74H,0BH,47H,90H
    DATA 90H,90H,0EBH,8,90H,86H,0C0H,86H
    DATA 0C0H,0EBH,1,90H,8AH,0E0H,0E2H,0E3H
    DATA 89H,3CH,0CBH
'-----------------------------------------------------------------
'
    CLS:CONST = 1.551868      'Calibration constant
'
' --- Set up input parameters to send to frequency subroutine ---
    LINE_IN$="LA42"     'Input line (any digital input line in main unit)
'
'
'--- Insert parameter values in RAM ---
    X=VAL RIGHT$(LINE_IN$,2)-41
    PORT=INT(X/8)                    'Port #
    L=INT((X/8-PORT)*8)              'Line#
    LINE=1:IF L=0 THEN GOTO L1
    FOR I=1 TO L:SHL LINE:NEXT        'Line Mask
'
L1:  POKE 17AH,PORT                  'Insert line address information in RAM
    POKE 17BH,LINE
'
    POKE 17CH,0:POKE 17DH,0          'Insert count time information in RAM
'
'
L0:  SYS PROG_SEG,0                  'Call frequency count subroutine
    CNT=PEEK 17FH*256 + PEEK 17EH   'Number of cycles during count duration
    FREQ=CNT*CONST                   'FREQ = frequency in Hz.
'
    CLS 2:DISP " CNT =";CNT
    CLS 3:DISP "FREQ =";FREQ
    GOTO L0
/
```

1041

SERIAL PORT EXPANDER

OPERATOR'S MANUAL

Y-tek, Inc.

Every effort has been made to ensure that this manual is error free. If you should find any errors, we would appreciate it if you would notify Y-tek, Inc. We would also appreciate any input or suggestions for improvement. Although every effort has been made to eliminate errors, Y-tek cannot assume responsibility for any errors in this manual or their consequences.

1041 Serial Port Expander Manual, First Edition 1994

Y-tek, Inc.

2634 Pleasant Union Church Road
Raleigh, North Carolina 27614
www.Y-tek.com
Telephone: (919) 676-4741
E-mail: info@Y-tek.com

Product Description

The 1041 Serial Port Expander is an optional accessory for the FT-100 Functional Tester that increases the number of serial communications ports from one to four. This allows the FT-100 to communicate with several devices without requiring cable changes. Each port of the 1041 is bidirectional and supports RTS/CTS handshaking, just like the FT-100. All necessary hardware and software needed to interface with the 1041 is already built into the FT-100. The 1041 is not designed to be controlled by any serial communications device other than the FT-100.

How the 1041 Works

The 1041 is a four-position serial port switch that is controlled by the FT-100. All data and handshake lines used by the FT-100 are switched. Once the "switch" is set, all serial communications are directed through the selected serial port. The selected serial port will remain the active port until a different serial port is selected.

The serial port may be switched several times within a program, but when the program ends, the 1041 is automatically switched back to serial port #1. Serial port #1 is always the default communications port. For this reason, it is recommended that data terminals or computers used to control the FT-100 be connected to this port. The following are conditions that cause the 1041 to automatically switch to serial port #1:

- Program Ends.
- Program is stopped (STOP pushbutton, or direct command)
- Program is placed in a "wait" state (WAIT pushbutton, or WAIT command)
- Error condition
- Single-Step Operation
- Reset (RESET button, or direct command)

Serial Port #1 is always the default port!

Port selection may be done through each of the following commands: PRINT, IN, or GET. Once a serial port has been selected, there is no way to later determine which serial port is selected.

How to use the 1041

Turn off the FT-100 power and connect the 1041 cable to the FT-100 serial communications port. Power may be turned back on after this connection is made. If a data terminal or computer is to be used to control the FT-100, it should be connected to port #1. All other serial devices may be connected to the other three ports. The port numbers are clearly marked on the front and back panels of the 1041.

The 1041 switches all hardware handshake lines as well as the data lines. Each device connected to the 1041 must be able to support RTS/CTS handshaking or have a shorting jumper between the RTS & CTS lines (pins 4 & 5). All hardware connections and specifications are the same as the FT-100. (See the FT-100 Operator's Manual for more information on cable requirements, serial communications protocol, and connector pinout).

The 1041 is now ready to provide communications with up to four serial devices. All functions of the 1041 are controlled by the FT-100 and are transparent to the operator.

1041 Control Commands

The 1041 is controlled by the PRINT, IN, and GET commands. If you are unfamiliar with these commands, see the FT-100 Operator's Manual. Following are brief descriptions of these commands and their interaction with the 1041:

IN -

Syntax: **IN** {serial port selector;}{string variable},{string variable}, ...

Notes: Program will stop and wait for an incoming character string through the active serial communications port. The active serial port may be switched by specifying the serial port#.

Examples:

```
IN @3;A$,B$,@2;C$     'Input character strings A$ and B$ from serial
                      'port# 3, then input character string C$ from
                      'serial port# 2
IN @4;A$;@2           'Input character string A$ from serial port#4,
                      'then switch the 1041 back to port #2
```

PRINT -

Syntax: **PRINT** {format;}{serial port selector;}{expression}{del}{expression}{del} .....

Notes: {format} and {serial port selector} may occur anywhere within the PRINT command. Number format and/or serial port may be changed several times within the same command. Once set, everything following in the command will use the last settings.

Examples:

```
PRINT @1;VAR1;@2;VAR2    'Print VAR1 out serial port#1, and VAR2
                         'out port #2
PRINT VAR1;@4;#B;VAR2    'Print VAR1 out last selected serial port#,
                         'then VAR2 out port #4 in Binary format.
```

GET -

Syntax: **GET** {serial port selector;}{string variable},{string variable}, ...

Notes: This command reads one character through the serial communicaitons port. The GET command treats incoming characters differently, depending on whether the Direct Command Mode is enabled or disable. For more information about enabling and disabling the Direct Command Mode, see the FT-100 Operator's Manual.

Direct Command Mode Enabled - When the program reaches the GET command, the serial input buffer is cleared and the program will wait for an incoming character. The first character is read in and placed in the referenced string variable.

Direct Command Mode Disabled - Incoming serial data goes into the serial input buffer in the order in which it is received (even as the program is running). The GET command will extract one character from the buffer, in the same order in which it was input (first in, first out). This is normally considered the best mode in which to use the GET command.

Examples:

```
GET @3;A$,B$,@2;C$    'Input two characters through port# 3. Put the
                      '1st into A$ & the 2nd into B$. Then input a
                      'character through serial port# 2 & place in C$.
```

1041 Control Command Notes:

As can be seen from the examples of these commands, the syntax for switching the 1041 is the same for each command.

{serial port selector;} consists of "@", followed by the desired serial port# (1-4), followed by a semicolon. If the port# is the last item on the command line, the semicolon may be omitted.

For all these commands, the active serial port may be switched anywhere within the statement. The serial port may be specified by a number (1-4), a variable, or a numerical expression. The selected port will remain until changed by another command, or the program ends.

Notes and Precautions

- Port #1 is always the default port. Whenever power is turned on, RESET issued, program started or stopped, WAIT command issued, or an error encountered, port #1 is automatically selected. Because of this, a data terminal or computer used to communicate with the FT-100 should normally be connected to port #1. This also applies when a terminal or computer is to be used in direct mode operation, for troubleshooting or program development, etc.

- The only way to switch the 1041 serial ports is through the IN, PRINT, or GET commands.

- Once set, the selected serial port will remain the active port until changed by another IN, PRINT, or GET command. The only exception to this is the default conditions that will automatically select port #1.

- If a serial port is not specified in the IN, PRINT, or GET command, the last selected port will be used.

- The serial port select syntax used in the IN, PRINT, or GET commands is identical and must be followed by a semicolon unless it is the last entity of the command line.
  Examples:
  ```
  IN @3;A$
  IN A$,B$,@4
  PRINT VAR1,@3;B$
  PRINT @2;A$;@3;VAR1;@1
  GET @4;A$,@2;B$
  ```

- Even though up to four serial devices may be connected to the 1041, the FT-100 can communicate with only one device at any given time. A device on an unselected port does not have access to the FT-100 until its port is selected by the FT-100.

- Serial communications protocol and hardware specifications for the 1041 are the same as the FT-100. See the FT-100 Operator's Manual for more information.

- After a WAIT command has been executed, or during the single-step pause, port #1 is automatically selected. This allows direct mode access during the "pause". When a CONT command is issued, the 1041 is changed back to its original port# and the program continues. WAIT and CONT commands may be issued via program commands or by pressing the WAIT/CONT push button.

- The serial communications port may be switched, without sending or receiving data, by either of the following commands:

```
PRINT @3;;      'Serial line will be switched, but nothing printed.
GET @2;A$       'For this command to work, and not get tied up, the Direct Command
                'Mode must be disabled.  A "null" character will be returned.
```

  Note: Do not use the IN command to switch the serial ports unless incoming data is expected. The program will wait for a character string, followed by a carriage return.

- If the 1041 is disconnected and reconnected after a port has been selected, the selected port will remain as the active port. (Note: Connecting or disconnecting the 1041 with power applied is not recommended)

- TXD, RXD, RTS, and CTS lines (pin numbers 2,3,4,5 respectively) are switched within the 1041. The shield and ground lines (pin numbers 1 & 7 respectively) are interconnected to all port connectors at all times. No other lines are used.

- The 1041 cannot be controlled by any serial device other than the FT-100.

- A second 1041 cannot be connected to a 1041 expansion port (to allow more than 4 serial ports). If more than four serial communications ports are needed, contact the factory.

- Each serial device connected to the 1041 must either support RTS/CTS handshaking or have these lines shorted together to bypass the handshaking requirement. See the FT-100 Operator's Manual for detailed information about the handshaking requirement.